Universidad
Carlos III de Madrid
www.uc3m.es

Trabajo fin de grado en Ingeniería electrónica Industrial y Automática

Diseño de un sistema tolerante a fallos basado en el procesador Nios II

Autor: David Lónchez López

Director: Luis Alfonso Entrena Arrontes

26 de Agosto de 2012

Índice

Índice de figuras

Agradecimientos

En primer lugar me gustaría mostrar mi agradecimiento a mi tutor de proyecto Luis. No sólo por aconsejarme y dirigir mis pasos cada vez que aparecía alguna dificultad durante todo el proceso, sino también por darme la oportunidad de participar en un proyecto de un interés formativo extraordinario para cualquier estudiante de ingeniería electrónica.

Ahora que finaliza una etapa tan importante en la vida de cualquier estudiante, no quiero olvidarme de todos los compañeros y amigos que me han acompañado durante todos estos largos años de estudio, sin los cuales estoy seguro que el camino hubiera sido mucho más duro.

Por último lugar, quiero expresar mi más profundo agradecimiento a mis padres, Óscar e Isabel, especialmente por todo su apoyo en los momentos difíciles. Por supuesto agradecer también su apoyo a mis dos hermanos, Raúl y Raquel, que siempre han estado ahí cuando los he necesitado.

1. Introducción

1.1 Introducción

Uno de los campos de mayor importancia en el desarrollo de nuevos sistemas electrónicos digitales es la fiabilidad. Un uso cada vez más extensivo de estos sistemas en aplicaciones de todo tipo y condición, está provocando que dicho campo sea una de la principal preocupación de los desarrolladores de hardware en los últimos tiempos.

Es cada vez más habitual encontrar este tipo de sistemas integrados en proyectos tecnológicos de gran envergadura en los cuales es frecuente que la vida, o la integridad de personas se encuentren en peligro dependiendo de un funcionamiento correcto de la electrónica incluida. Como claros ejemplos de lo mencionado podemos encontrar sistemas utilizados en medicina o en la automoción.

Otra de las razones fundamentales para explicar este aumento de la preocupación sobre la fiabilidad se puede encontrar en desarrollos que necesitan un coste elevado y donde un fallo no esperado puede suponer la pérdida de grandes cantidades de dinero y recursos invertidos, caso evidente de la industria aeroespacial.

Existen varios tipos de fallos que pueden afectar a un circuito electrónico digital:

- Transitorio (*soft error*). Cambio aleatorio y puntual de alguna de las señales del circuito debido a la incidencia de radiación electromagnética o la colisión de partículas de alta energía, con la energía suficiente como para cambiar el estado del circuito.

- Permanente (*hard error*). Fallo perpetuo debido a la rotura o deterioro de un componente electrónico del circuito.

- Intermitente. Fallo que se produce de forma periódica en el mismo punto de algún cierto componente. Por lo general es indicativo de un fallo permanente en el futuro próximo.

Este proyecto se centra en el desarrollo tolerante a fallos transitorios de un sistema basado en un procesador comercial del fabricante Altera, el Nios II. Estos fallos constituyen un problema de importancia en aumento, especialmente los debidos a la radiación. Con la disminución del tamaño de la electrónica digital, aparte de abaratar su coste se consigue que la energía de activación de los componentes sea menor. Esto hace que su sensibilidad sea mayor y que por tanto sea necesaria menor energía tanto para modificar su estado como para que se produzcan fallos. Hasta tal punto es así que hoy día muchos dispositivos electrónicos, y en especial las memorias, presentan niveles inaceptables de sensibilidad a la radiación incluso a nivel terrestre.

Es enormemente difícil y extremadamente costoso evitar que se produzcan errores en un sistema cualquiera, así que la forma de aumentar la fiabilidad pasa por el desarrollo de sistemas tolerantes a fallos. Estos sistemas son capaces, en mayor o menor medida, de proseguir con su funcionamiento habitual aún en presencia de errores. Existen varias formas de implementar un sistema tolerante a fallos:

- La solución más directa es la replicación de todo el sistema al completo, decidiendo mediante un sistema de votación la respuesta válida tras la ejecución de cada una de las rutinas realizadas. Pese a ser una solución evidente y de fácil implementación, su coste es elevado, especialmente cuando nos encontramos en desarrollos de alto nivel.

- Una solución más limitada en coste que la anterior es la replicación únicamente del componente del que se quiere hacer tolerante a fallos. Pese a ser una solución más barata que la anterior, su coste sigue siendo elevado.

- Una tercera manera es el fortalecimiento a nivel de sistema. Esta solución consiste en la modificación del hardware y/o del software para aportar al sistema robustez frente a fallos. Se trata de una buena solución si tiene un coste moderado y se obtiene una fiabilidad aceptable.

Es este último enfoque el adoptado en este proyecto. En concreto, se opta por una solución híbrida que combina modificaciones de software con la inserción de un I-IP (*Infrastructure IP*) previamente desarrollado para un sistema basado en un microprocesador LEON3. Los I-IP son módulos IP que añaden propiedades no funcionales, como por ejemplo fiabilidad, al sistema al cual se conectan.

1.2 Objetivos

El objetivo del presente proyecto, es la adaptación de un módulo IP que se conecte al bus de comunicaciones Avalon de un sistema basado en un microprocesador Nios II de forma no intrusiva, y que con las correspondientes modificaciones del software ejecutado sea capaz de detectar errores transitorios en el sistema de forma no intrusiva. La solución propuesta debe ser eficiente, es decir, debe de ser capaz de detectar una proporción de errores lo más elevada posible, afectando mínimamente a las prestaciones del procesador. Dicho objetivo se evaluará según su ocupación en la FPGA tras su implementación.

1.3 Fases del desarrollo

Para la realización del presente proyecto se parte de conocimientos previamente adquiridos en las asignaturas de Fundamentos de ingeniería electrónica, Programación, Informática industrial, Electrónica digital, Diseño de circuitos integrados y Microprocesadores. A continuación se detallan las distintas fases del desarrollo del proyecto.

En primer lugar se estudió la arquitectura del microprocesador Nios II, así como los estándares característicos del protocolo de comunicaciones Avalon.

Debido a la necesidad de adaptación de un sistema previo basado en otra arquitectura e interfaz de bus diferente, también se revisó la arquitectura del microprocesador LEON3 y el estándar del bus AMBA.

A continuación se llevaron a cabo diversos diseños de prueba siguiendo tutoriales recomendados por el propio fabricante para familiarizarse con las distintas herramientas a utilizar durante el proyecto.

Una vez adquiridos todos los conocimientos necesarios para poder comenzar con el diseño, se procedió a rediseñar el módulo IP creado anteriormente mediante la renovación de sus señales y la readaptación de su lógica a los nuevos interfaces. Esto supuso numerosas versiones verificadas mediante simulación y optimizadas hasta llegar a la versión final.

En último lugar se han efectuado las diferentes pruebas de validación; por un lado se realizó una síntesis del código VHDL del módulo de detección para conocer los requisitos de espacio y velocidad de funcionamiento, y por otro se efectuó una serie de test de errores para comprobar la capacidad de detección de fallos del módulo diseñado.

1.4 Medios empleados

En la realización de este proyecto se ha utilizado un ordenador con sistema operativo Windows 7, además de la siguiente lista de software especializado:

- Quartus II:

Quartus II es una herramienta desarrollada por el fabricante Altera para el análisis y la síntesis de diseños de sistemas digitales implementados sobre FPGAs.

Permite el desarrollo de sistemas utilizando lenguajes HDL o bloques lógicos. Para sistemas basados en el microprocesador Nios II permite la instanciación del componente diseñado en Qsys para incorporar lógica externa implementada dentro de la FPGA.

- Qsys:

Qsys es una herramienta de integración de sistemas que forma parte del software Quartus II. Permite el diseño de sistemas basados en el microprocesador Nios II gracias a un conjunto de módulos predefinidos adaptados al sistema del bus Avalon.

También permite la incorporación de módulos de diseño propio programados en lenguaje HDL para personalizar los diversos periféricos.

- Nios II Software Build Tools for Eclipse:

Nios II SBT es una plataforma de desarrollo integrado especialmente preparada para la programación de los sistemas basados en dicho procesador. Se encuentra totalmente integrada en el paquete de herramientas proporcionado por la propia Altera, lo que facilita su uso durante el desarrollo del sistema completo.

- Modelsim:

Modelsim es un software que administra un entorno que permite editar, compilar, simular y depurar diseños de sistemas digitales descritos en VHDL o Verilog. La propia Altera ofrece una versión personalizada para sus propios dispositivos denominada Modelsim-Altera que permite compilar de manera automática todas las librerías necesarias para la simulación de los sistemas desarrollados mediante las herramientas descritas anteriormente.

1.5 Estructura de la memoria

A continuación se resume brevemente el contenido de los capítulos en los que se divide esta memoria.

En el capítulo 1 (*Introducción*) se describe el contexto en el que se enmarca el presente proyecto, la problemática que se pretende resolver y los medios empleados para ello, así como un breve resumen del proceso de ejecución del proyecto al completo.

En el capítulo 2 (*Estado del arte*) se exponen de forma resumida las bases y herramientas sobre las que se asienta el presente proyecto: el protocolo de comunicaciones Avalon, el microprocesador Nios II y las técnicas de detección de errores que tienen relación con el presente proyecto.

El capítulo 3 (*Diseño del módulo de detección de fallos*) contiene una descripción detallada del dispositivo diseñado.

El capítulo 4 (*Diseño del sistema Nios II*) explica los pasos seguidos para la configuración del sistema basado en el microprocesador Nios II, así como la implementación en él del módulo de detección de fallos diseñado.

En el capítulo 5 (*Resultados*) se explican las distintas pruebas realizadas para determinar las características del módulo IP diseñado y se muestran los resultados obtenidos en dichas pruebas.

En el capítulo 6 (*Conclusiones*) se infieren una serie de conclusiones lógicas a partir de los resultados experimentales obtenidos.

2. Microprocesador Nios II y Bus Avalon para FPGAs de Altera

2.1 Microprocesador embebido Nios II para FPGAs de Altera

El Nios II es un núcleo procesador configurable proporcionado por el fabricante Altera para ser utilizado sobre sus FPGAs comerciales [1]. Se trata de un procesador RISC de 32 bits de propósito general basado en una arquitectura tipo Harvard, (usa buses separados para instrucciones y para datos). Entre sus características principales encontramos:

- Juego completo de instrucciones, datos y direcciones de 32 bits.
- 32 registros de propósito general.
- Juegos opcionales de registros shadow.
- 32 fuentes de interrupción externa.
- Una interfaz de control de interrupciones externas para fuentes adicionales.
- Instrucciones dedicadas para multiplicaciones de 64 y 128 bits.
- Instrucciones para operaciones de coma flotante en precisión simple.
- Operaciones de multiplicación y división de 32 bits.
- Acceso a variedad de periféricos integrados e interfaces para el manejo de memorias y periféricos externos.
- Una memory management unit (MMU) opcional para soportar sistemas operativos que la necesiten.
- Una memory protection unit (MPU) opcional.
- Entorno de desarrollo de software basado en la herramienta GNU C/C++.

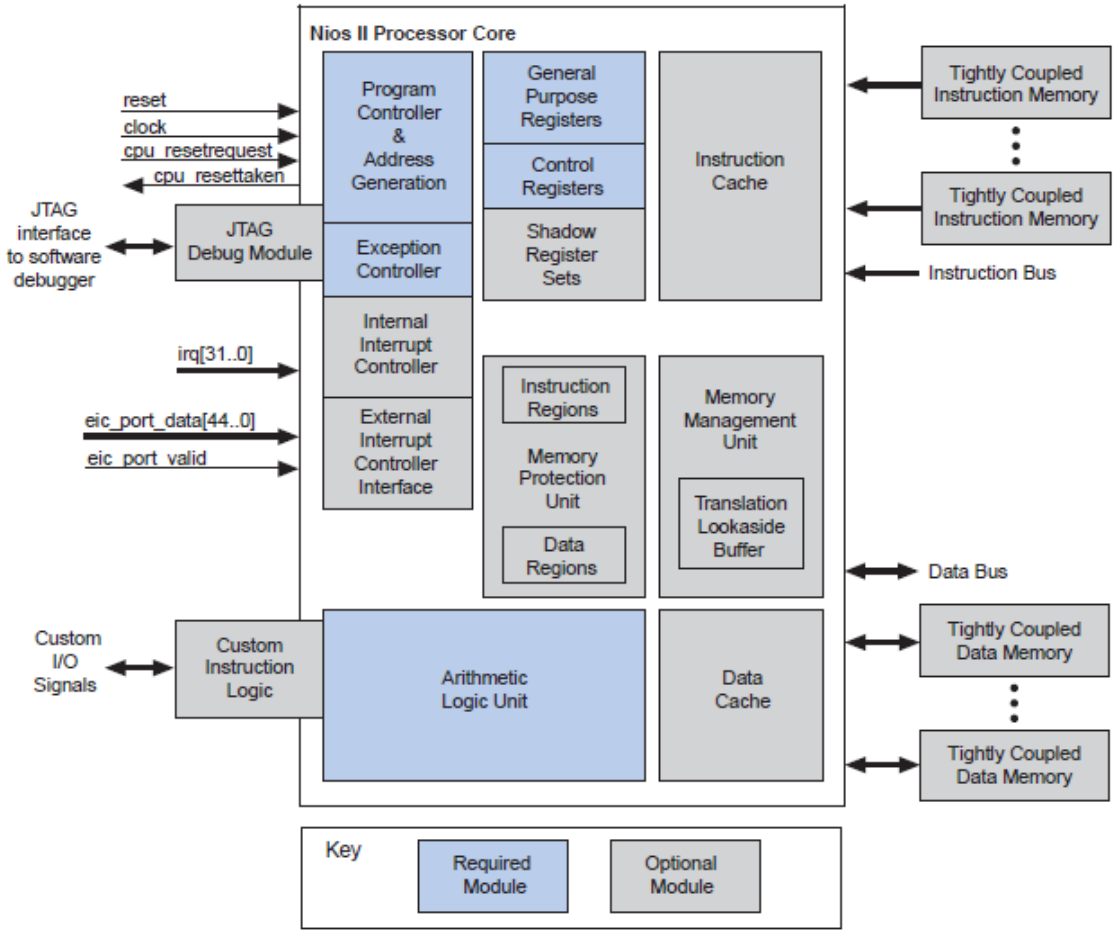

Figura 1: Diagrama de bloques del núcleo del procesador Nios II [1]

2.1.1 Unidad Aritmético lógica

La ALU del Nios II opera con datos almacenados en registros de propósito general. Las operaciones de la ALU reciben uno o dos datos de entrada desde los registros y almacena el resultado en otro.

Las operaciones soportadas por la ALU del Nios II son:

Categoría	**Descripción**
Aritméticas	*Suma, resta, multiplicación y división de operandos con y sin signo*
Relacionales	*Operaciones de relación entre operandos con y sin signo del tipo; Igual, no igual, mayor o igual que, menor que.*
Lógicas	*AND, OR, NOR y XOR*
Rotación y desplazamiento	*Operaciones de rotación y desplazamiento de datos.*

2.1.2 Gestión de excepciones e interrupciones

El procesador Nios II incluye hardware para el manejo de excepciones, incluyendo interrupción hardware. También incluye una interfaz de control de interrupciones externas opcional (EIC).

La arquitectura del Nios II proporciona un controlador de excepciones muy simple para manejar todo tipo de ellas. Cada excepción, incluyendo las interrupciones internas de hardware, causa que el procesador transfiera la ejecución mediante una dirección de excepción. En dicha dirección, un controlador de excepciones determina la causa de la excepción y ejecuta la rutina correspondiente.

Estas direcciones de excepción se pueden comprobar en el propio software de configuración del sistema Nios II, permitiendo así un control óptimo del diseño.

Todas las excepciones son precisas. Esto quiere decir que el procesador tiene que completar la ejecución de todas las instrucciones precedentes a la causa que la genera, y no comenzará la ejecución de las instrucciones posteriores al momento de la generación de dicha interrupción hasta finalizar la rutina correspondiente.

En el sistema Nios II encontramos dos tipos diferentes de excepciones:

- Excepciones software: producida al aparecer en un programa una instrucción de interrupción que transfiriere el control a un programa diferente.

- Excepciones hardware: cuando se produce un evento sobre cualquiera de las 32 entradas de petición de interrupción disponibles en el microprocesador (IRQ0 a IRQ31).

El controlador de interrupciones externo proporciona un control de interrupciones vectorizado (VIC, Vectored Interrupt Controller) con el que se da respuesta a las interrupciones a través de vectores de interrupción separados y según un determinado nivel de prioridad. Así, cuando se produce una interrupción se transfiere la ejecución directamente a la rutina de atención (ISR, Interrupt Service Rutine) apropiada, indicada por dicho vector a través de una instrucción de salto.

Hay que tener en cuenta que, para que se pueda generar una interrupción, se deben cumplir las siguientes tres condiciones:

- Que se encuentre activado el bit (PIE) del registro de estado que habilita las interrupciones de forma global.
- Que se produzca una llamada a una de las entradas de atención de interrupción (IRQ).

- Que se encuentre activado el bit correspondiente a dicha entrada en el registro de control de interrupciones habilitadas (IENABLE).

2.1.3 Conexión y acceso a memoria y periféricos

El núcleo del Nios II utiliza uno o más de los siguientes modos de conexión para acceder a memoria o a los periféricos:

- Puerto maestro de instrucciones y datos: Un puerto maestro del tipo Avalon Memory-Mapped(Avalon-MM) que conecta las instrucciones o los datos almacenados en memoria mediante un sistema de interconexión por defecto para los sistemas Nios II.
- Conexión a través de caché (para datos e instrucciones): Una memoria caché interna y de acceso rápido.
- Conexión directa a bloques de memoria externos al propio núcleo del Nios II.

La propia arquitectura del Nios II oculta los detalles de configuración del hardware al diseñador, lo que permite un diseño sencillo sin necesidad de conocimientos específicos para su implementación.

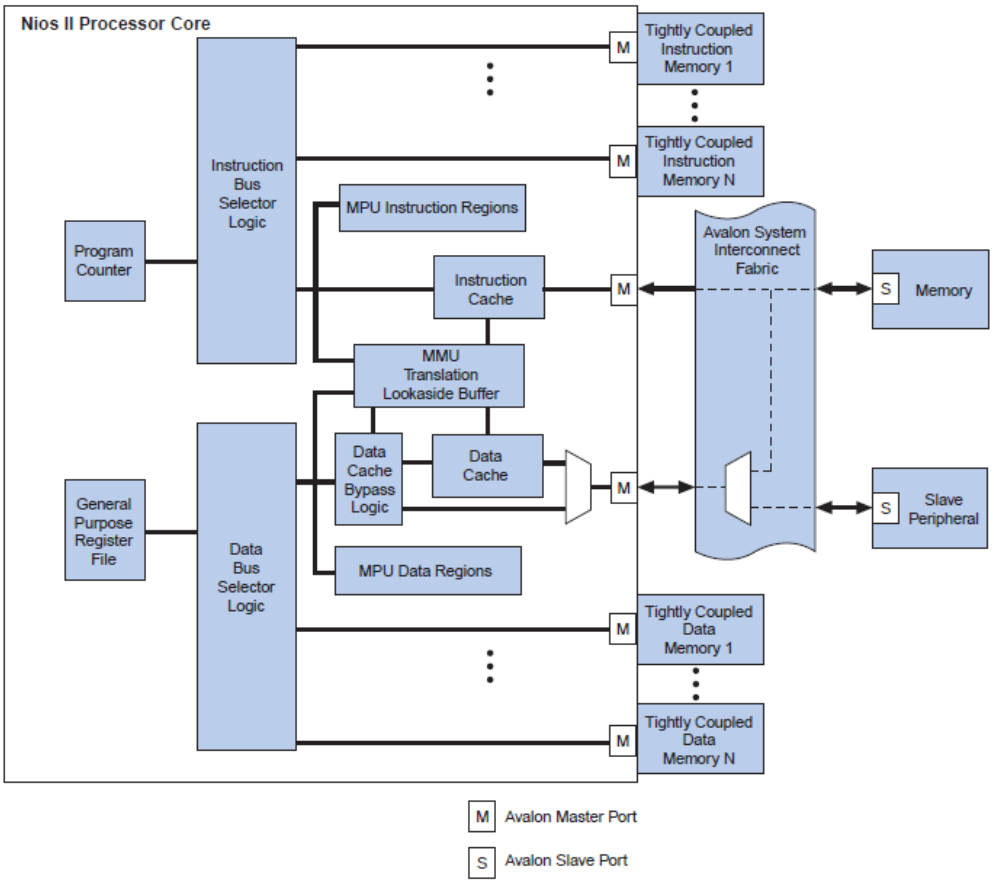

Figura 2: Organización de la memoria y periféricos del Nios II [1]

La arquitectura del Nios II proporciona buses separados para las instrucciones y los datos, (tipo Harvard). Ambos buses están implementados bajo las especificaciones del sistema Avalon-MM. Los puertos maestros de los datos se conectan tanto a las memorias como a los periféricos, mientras que los de instrucciones se conectan únicamente a las memorias.

El bus de instrucciones del Nios II esta implementado como un puerto maestro de 32 bits del tipo Avalon-MM. Este puerto realiza una única función: recoge las instrucciones para ser ejecutadas por el procesador. Los puertos de instrucciones no realizan ningún tipo de operación de escritura.

Este puerto maestro es del tipo pipelined Avalon-MM. Esta configuración minimiza el impacto de transferencias síncronas de memoria e incrementa la frecuencia máxima general de todo el sistema. Este maestro puede emitir peticiones de lectura sucesivas antes que el dato de peticiones previas haya sido registrado.

Este maestro siempre devuelve datos de 32 bits, utilizando un sistema lógico dinámico para adecuar el ancho del bus contenido en los propios sistemas por defecto de interconexión. Gracias a ello, cada instrucción recogida devuelve una palabra completa independientemente del ancho de la memoria objetivo.

El bus de datos del Nios II esta implementado como un puerto maestro de 32 bits del tipo Avalon-MM. Dicho maestro realiza dos funciones:

- Lectura de datos desde un dispositivo de memoria o un periférico cuando el procesador ejecuta una instrucción de carga.
- Escritura de datos a un dispositivo de memoria o periférico cuando el procesador ejecuta una instrucción de almacenamiento.

La señal de Byteenable en el maestro especifica cuáles de los cuatro bytes que forman el valor total del dato deben de ser utilizados durante las transferencias de escritura.

La arquitectura del Nios II proporciona memorias caché tanto para los maestros de instrucciones como para los datos. Esta memoria caché reside dentro del propio núcleo del procesador Nios II como parte integral de éste. Este tipo de memorias permite la mejora del tiempo medio de acceso para los sistemas basados en Nios II que usan dispositivos de memoria externos lentos tales como un SDRAM.

Las cachés de datos e instrucciones están habilitadas de manera permanente durante la ejecución del programa, pero se proporcionan métodos para el software con el objeto de evitar que los accesos a los periféricos no devuelvan datos a la caché. La gestión de la caché y su correcto funcionamiento está manejado por software.

Estas memorias caché son opcionales y dependen directamente del objetivo del diseño a realizar. El núcleo de un procesador Nios II puede incluir una, ambas o ninguna de las memorias caché. Y en caso de ser usadas, ambas son completamente configurables por el diseñador. Su uso no afecta a la funcionalidad de los programas cargados, sin embargo sí que afecta a la velocidad a la que el procesador ejecuta las instrucciones y la lectura/escritura de los datos.

Existen dos métodos para la derivación de la caché:

- Instrucciones de carga y almacenamiento para periféricos I/O: Instrucciones como ldio o stio derivan la caché de datos y fuerzan una transferencia de datos a una dirección específica.
- Método de derivación caché del bit-31: este procedimiento usa el bit 31 de la dirección como un marcador para indicar si el procesador debería transferir datos a/desde una caché, o bien derivarlos.

La conexión directa a bloques de memoria externos garantiza accesos de baja latencia para aplicaciones que lo necesiten. Entre los beneficios de este tipo de accesos se encuentran:

- Un rendimiento similar a la memoria caché.
- El software puede garantizar que el código o los datos críticos se encuentran localizados en el dispositivo de memoria externo.

- No existe sobrecarga en el almacenamiento a tiempo real, tales como carga, invalidación o borrado de memoria.

El mapa de direcciones para memoria y periféricos en un sistema basado en un procesador Nios II depende cada diseño. Dicho mapa de direcciones se especifica antes de generar el sistema.

Hay tres direcciones que son parte del procesador y merecen una mención especial:

- La dirección de reset.
- La dirección de excepción.
- La dirección de ruptura del controlador.

La unidad de gestión de memoria opcional presente en los diseños basados en el procesador Nios II proporciona, entre otras, las siguientes funcionalidades:

- Mapeado virtual para direcciones físicas.
- Protección de memoria.
- Direcciones virtuales y físicas de 32 bits.
- Un espacio de 512 MB de direcciones físicas disponible para accesos directos.
- Hardware translation lookaside buffers (TLBs), que permite la aceleración de las traducciones de las direcciones:
    - TLBs separados para accesos a instrucciones y datos.
    - Lectura, escritura y ejecución de permisos controlados por sección.
- Formato de tablas de sección (o estructuras de datos equivalente) determinado por software.

La unidad de protección de memoria opcional del Nios II proporciona las siguientes funcionalidades:

- Protección de memoria.
- Hasta 32 regiones de instrucciones y otras 32 de datos.
- Tamaños variables de región para datos e instrucciones.
- Cantidad de región de memoria definida por tamaño o por límite superior de dirección.
- Permisos de acceso a lectura y escritura para las regiones de datos.
- Permisos de acceso a ejecución para las regiones de instrucciones.
- Regiones de solapamiento.

Los sistemas basados en Nios II pueden incluir bien un MPU o un MMU, pero no pueden incluir a ambos a la vez en el mismo núcleo del procesador.

2.1.4 Módulo JTAG debug

La arquitectura del Nios II proporciona un módulo JTAG debug que facilita funcionalidades emuladas para controlar el procesador de manera remota desde un PC. Entre todas las características de este módulo están:

- Programas descargables a la memoria.
- Comienzo y parada de ejecución.
- Establecimiento de breakpoints y watchpoints.
- Análisis de registros y memoria.
- Recogida de taza de datos de ejecución a tiempo real.

Las sondas externas de debbuging pueden acceder al procesador a través de la interfaz JTAG estándar en la FPGA. En el lado del procesador, el módulo de debug se conecta a las señales internas del núcleo del procesador. Este módulo tiene un control no enmascarable sobre el propio procesador, y no requiere código auxiliar de software relacionado con la aplicación que se está sometiendo a prueba. Todos los recursos del sistema visibles para el procesador en modo supervisión están disponibles para el módulo de debug. Para la recolección de la traza de datos, el módulo almacena dicha traza en memoria o en la propia sonda de debug.

El módulo de debug obtiene el control del procesador bien mediante la activación de señales de hardware break, o bien mediante la escritura de instrucciones tipo break dentro del programa cargado en memoria que va a ser ejecutado. En ambos casos, el procesador transfiere la ejecución a la rutina localizada en la dirección de ruptura.

Los software breakpoints permiten establecer breakpoints en las instrucciones incluidas en la RAM. Este mecanismo escribe una instrucción de ruptura dentro del código ejecutable almacenado en la RAM. Cuando el procesador ejecuta dicha instrucción, el control se transfiere al módulo JTAG debug.

Los hardware breakpoints permiten establecer breakpoints en las instrucciones incluidas en memorias no volátiles, tales como una memoria flash. Este mecanismo monitoriza continuamente las direcciones de instrucción del procesador. Si la dirección de la instrucción coincide con la dirección del hardware breakpoint, le módulo de JTAG debug toma control del procesador.

La captura de traza hace referencia a la posibilidad de grabar la ejecución instrucción a instrucción del procesador mientras se ejecuta el código a tiempo real. El módulo JTAG debug proporciona las siguientes funcionalidades:

- Captura de las trazas de ejecución (ciclos del bus de instrucciones).
- Captura de traza de datos (ciclos del bus de datos).

- Para cada uno de los ciclos del bus de datos, captura de direcciones, datos, o ambos.
- Inicio y parada de captura de trazas a tiempo real, basada en disparadores.
- Inicio y parada manual de raza bajo el control del host.
- Parada opcional de captura de traza cuando el buffer de traza está lleno, dejando al procesador en ejecución.
- Almacenamiento de la traza de datos en buffer de memoria interna dentro del módulo JTAG debug. (Esta memoria es únicamente accesible mediante la conexión JTAG).
- Almacenamiento de la traza de datos para buffers amplios en una sonda de debug externa.

El módulo JTAG debug proporciona trazabilidad para el bus de instrucciones (traza de ejecución), el bus de datos (traza de datos), o ambos simultáneamente. La traza de ejecución únicamente graba las direcciones de las instrucciones ejecutadas permitiendo analizar donde se ejecutó el código dentro de la memoria. La traza de datos guarda los datos asociados a cada operación de carga o almacenamiento del bus de datos.

El módulo JTAG debug puede filtrar la traza del bus de datos a tiempo real para capturar los siguientes valores:

- Únicamente direcciones de carga.
- Únicamente direcciones de almacenamiento.
- Ambos tipos de dirección.
- Únicamente datos de carga.
- Direcciones y datos de carga.
- Direcciones y datos de almacenamiento.
- Direcciones y datos para carga y almacenamiento.

2.1.5 Juego de instrucciones

El procesador Nios II proporciona un juego de instrucciones de tipo RISC con una longitud de instrucción de 32 bits.

Dentro del conjunto de instrucciones podemos encontrar diez categorías:

- Instrucciones de carga y almacenamiento: este tipo de instrucciones son las encargadas de realizar las transferencias de contenido entre las memorias o las interfaces de entrada/salida y los registros de propósito general. Esto permite la lectura o escritura a distintos niveles de palabras, (32,16 y 8 bits).

- Instrucciones aritméticas: Este tipo de instrucciones realizan las operaciones aritméticas básicas con los datos presentes en los registros de propósito general o con un valor inmediato proporcionado por la instrucción.
- Instrucciones lógicas: Este tipo de instrucciones realizan las operaciones lógicas habituales con los datos presentes en los registros de propósito general o bien con un valor dado.
- Instrucciones de transferencia entre registros: Este tipo de instrucciones se encargan de copiar el contenido de un registro en otro o bien un valor indicado de forma inmediata.
- Instrucciones de comparación: Este tipo de instrucciones comparan el contenido de dos registros o el de un valor dado y el de un registro, y escriben el resultado en el registro designado para ello.
- Instrucciones de rotación y desplazamiento: Este tipo de instrucciones rotan o desplazan el contenido presente en un registro.
- Instrucciones de ruptura y salto: Este tipo de instrucciones se encargan de cambiar el flujo de ejecución de un programa bien de forma condicional o no.
- Instrucciones de llamada a subrutina: Este tipo de instrucciones permiten la llamada y el retorno de subrutinas.
- Instrucciones de manejo de excepciones: Este tipo de instrucciones generan una interrupción de software y el retorno de la rutina de atención de interrupción.
- Instrucciones de control: Este tipo de instrucciones especiales se utilizan para leer y escribir en los registros de control.

2.2 Bus Avalon

La familia de interfaces Avalon define unos protocolos adecuados para la transmisión de datos de alta velocidad, lectura y escritura de registros y memoria, y control de dispositivos externos. Todas estas interfaces están incorporadas en los distintos componentes disponibles en la herramienta Qsys.

Existen siete tipos de interfaces disponibles:

- Avalon Streaming Interface (Avalon-ST): Una interfaz que proporciona un flujo de datos unidireccional, incluyendo streams multiplexados, paquetes y datos DSP.
- Avalon Memory Mapped Interface (Avalon-MM): Una interfaz de lectura/escritura basada en direcciones entre esclavos y maestros.
- Avalon Conduit Interface: Una interfaz que reúne señales individuales o grupos de ellas que no encajan en el resto de tipos de señales del sistema Avalon. Permite conectar interfaces de tipo Conduit dentro de un sistema Qsys o bien exportarlos para conectarlas con módulos externos al diseño o a pines de la FPGA.

- Avalon Tri-State Conduit Interface (Avalon-TC): Esta Interfaz proporciona conexiones para periféricos externos. Esto permite que múltiples periféricos puedan compartir pines a través de una señal multiplexada, reduciendo la cantidad de pines utilizados de la FPGA y el número de trazas del PCB.
- Avalon Interrupt Interface: Esta interfaz permite a los componentes señalar eventos a otros componentes.
- Avalon Clock Interface: Una interfaz que conduce o recibe relojes, (todas las interfaces Avalon son síncronas).
- Avalon Reset Interfaces: Esta interfaz proporciona la conectividad de la señal reset.

Un solo componente puede incluir cualquier número de todas estas interfaces y también puede incluir múltiples instancias del mismo tipo de interfaces.

Todo el contenido de este capítulo se ha obtenido del documento de especificación Avalon [5], en el cual se exponen con detalle todas las características del bus.

2.2.1 Interfaces de reloj y reset

La interfaz de reloj del sistema Avalon, define el reloj o los relojes utilizados por un componente. Todos los componentes pueden tener clock inputs, clock outputs, o ambos.

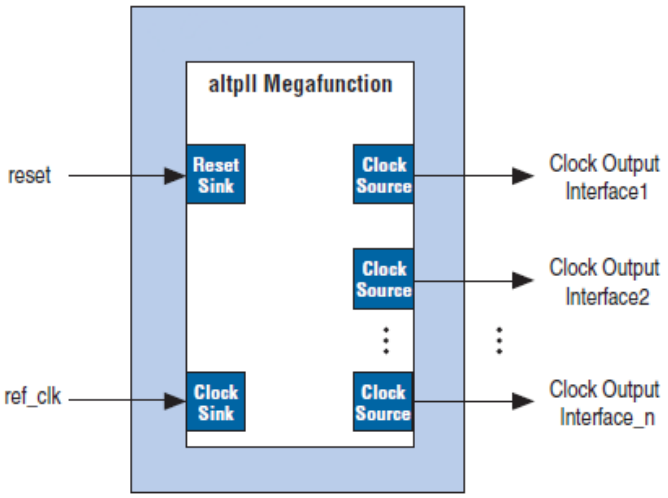

Figura 3: Ejemplo de interfaz de clock inputs y outputs [5]

Cualquier componente típico de un diseño basado en el sistema Avalon incluye una fuente de reloj para proporcionar una referencia temporal a otras interfaces y lógica interna.

- **Clock Sink**

Roles de las señales del Clock Sink				
Rol de la señal	***Anchura***	***Dirección***	***Requerida***	***Descripción***
Clk	1	Input	Sí	Señal de reloj. Proporciona sincronización para la lógica interna y para otras interfaces

Propiedades del Clock Sink			
Nombre	***Valor por defecto***	***Valores permitidos***	***Descripción***
clockRate	0	0 - (2^{32}-1)	Indica la frecuencia en Hz de la interfaz del clock sink. Si su valor es 0, la frecuencia no es relevante

Todas las interfaces síncronas tienen una propiedad denominada associatedClock que especifica que clock input del componente se usa como referencia de sincronización para la interfaz.

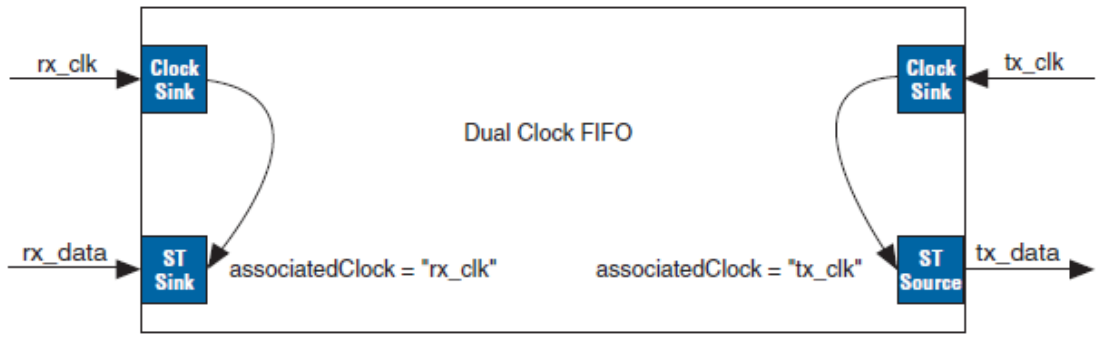

Figura 4: Propiedad associatedClock [5]

- **Reset Sink**

La interfaz de reset del sistema Avalon proporciona tanto funcionalidades de reset lógico que reinicia los registros y las memorias sin necesidad de apagar el dispositivo como la inicialización lógica del dispositivo después de su encendido.

Roles de las señales del Reset Input				
Rol de la señal	***Anchura***	***Dirección***	***Requerida***	***Descripción***
reset *reset_n*	1	Input	Sí	Resetea la lógica interna de una interfaz o componente a un estado definido por el usuario. Sincroniza el clock input en la interfaz del reloj asociado.

Propiedades de la interfaz de reset			
Nombre	***Valor por defecto***	***Valores permitidos***	***Descripción***
associatedClock	---	Nombre válido de un reloj	El nombre de un reloj que sincroniza esta interfaz.
synchronousEdge	DEASSERT	NONE DEASSERT BOTH	Indica el tipo de sincronización requerida por el reset input. ▪ NONE - No se requiere sincronización porque los componentes incluyen lógica para sincronización interna de la señal de reset. ▪ DEASSERT - La activación del reset es asíncrona y la desactivación es síncrona. ▪ BOTH - La activación y la desactivación son síncronas.

2.2.2 Interfaz Avalon Memory-Mapped

Las interfaces Avalon Memory-Mapped (Avalon-MM) se utilizan para las interfaces de escritura y lectura sobre componentes maestros y esclavos en un sistema memory-mapped. Estos componentes incluyen microprocesadores, memorias, UARTs, DMAs, y timers que presentan interfaces maestras y esclavas conectadas mediante una interconexión por defecto.

Los componentes del tipo Avalon-MM incluyen únicamente las señales necesarias para la lógica correspondiente incluida en dicho componente. Cada señal en un esclavo Avalon-MM corresponde a un único rol de señal del Avalon-MM. Un puerto del tipo Avalon-MM sólo puede usar una instancia de cada tipo de rol para las señales utilizadas.

2.2.2.1 Señales

Las señales disponibles para la interfaz Avalon-MM permiten crear maestros que usen modo ráfaga tanto para escritura como para lectura. No es necesario la presencia de todas las señales para definir una interfaz Avalon-MM. Los requisitos mínimos son readdata para una interfaz de sólo lectura o writedata para una interfaz de sólo escritura.

Todas las señales Avalon se activan a nivel alto. Las señales que puedes activarse también a nivel bajo muestran la posibilidad mediante la incorporación de la señal _n correspondiente.

Rol de señal	Anchura	Dirección	Descripción
Señales principales			
address	1-32	Maestro → Esclavo	Para los maestros, la señal address representa una dirección de byte. El valor de la dirección debe de estar alineado con la anchura del dato. Para escribir bytes específicos dentro de una palabra de datos, el maestro debe de utilizar la señal byteenable. Para los esclavos, la interconexión traduce la dirección de byte en una dirección de palabra dentro del espacio de direcciones del esclavo de manera que cada acceso se produzca como una palabra de datos desde la perspectiva del esclavo.
begintransfer	1	Maestro → Esclavo	Activada por la interconexión durante el primer ciclo de cada transferencia sin tener en cuenta la señal de waitrequest o cualquier otra señal. Si esta señal no se incluye en la interfaz Avalon-MM del maestro, Qsys la genera automáticamente.
byteenable *byteenable_n*	1,2,4,8, 16,32, 64,128	Maestro → Esclavo	Especifica la línea de byte durante las transferencias en puertos de anchura mayor a 8 bits. Cada bit de la señal byteenable corresponde a un byte en las señales writedata y readdata. El bit maestro <n> de la señal byteenable indica si el byte <n> está siendo escrito. Durante las escrituras, byteenable especifica que bytes están siendo escritos, haciendo que el resto de los bytes de la señal sean ignorados. Durante las lecturas, la señal byteenable indica que bytes son leídos por el maestro. Si una interfaz no presenta la señal

			byteenable, la transferencia se ejecuta como si todos los bits de la señal byteenable estuvieran activados. Cuando más de un bit de la señal byteenable está activado, todas las líneas activadas son adyacentes. El número de líneas adyacentes deben de ser una potencia de 2, y los bytes especificados deben de estar alineados con el límite de direcciones para el tamaño de los datos. Los posibles valores de la señal son: • 1111 escribe los 32 bits al completo • 0011 escribe los 2 bytes más bajos • 1100 escribe los 2 bytes más altos • 0001 escribe únicamente el byte 0 • 0010 escribe únicamente el byte 1 • 0100 escribe únicamente el byte 2 • 1000 escribe únicamente el byte 3
chipselect *chipselect_n*	1	Maestro → Esclavo	Cuando está presente, un puerto esclavo ignora todas las señales del interfaz Avalon-MM a menos que la señal chipselect esté activada. La señal chipselect debe ser utilizada en combinación con las señales read y write. No es una señal necesaria.
debugaccess	1	Maestro → Esclavo	Cuando está activada, permite escribir en las memorias internas que normalmente están protegidas contra escritura.
read *read_n*	1	Maestro → Esclavo	Cuando se activa permite una transferencia del tipo lectura. Siempre que se incluya dentro de una interfaz es obligatorio incorporar la señal readdata.
readdata	8,16,32, 64,128, 256,512, 1024	Esclavo → Maestro	La señal readdata transfiere los datos desde el esclavo hasta el maestro en respuesta a una activación de la señal de lectura del tipo read.
write *write_n*	1	Maestro → Esclavo	Cuando se activa permite una transferencia del tipo escritura. Siempre que se incluya dentro de una interfaz es obligatorio incorporar la señal writedata.
writedata	8,16,32, 64,128, 256,512, 1024	Maestro → Esclavo	La señal writedata transfiere los datos destinados a la escritura. La anchura de la señal debe ser la misma que la señal readdata si ambas se encuentran presentes en la interfaz definida.

Señales de estado de espera			
Lock	1	Maestro → Esclavo	La señal lock asegura que un maestro consigue la conexión mediante el árbitro interno del sistema, manteniendo el acceso al esclavo para transacciones múltiples. Se activa de manera simultánea con las primeras activaciones de las señales de read o write de una secuencia cerrada de transacciones, y se desactiva al final de la transacción de la secuencia. La activación de la señal lock no garantiza obtener la preferencia del árbitro correspondiente, pero después de su activación el maestro debe de mantener el acceso hasta que se desactive. Un maestro que incluya la señal lock no puede ser de tipo burst.
waitrequest *waitrequest_n*	1	Esclavo → Maestro	Activada por el esclavo cuando es incapaz de responder ante una petición de lectura o escritura mediante las señales read o write correspondientes. Fuerza al maestro a permanecer a la espera hasta que la interconexión esté preparada para proceder con la transferencia. Al comienzo de todas las transferencias, un maestro inicia la transferencia y espera hasta que la señal de waitrequest se desactiva. La señal waitrequest puede activarse a nivel alto o bajo, dependiendo de las propiedades del sistema. Cuando la señal waitrequest se activa, las señales de control del maestro permanecen constantes con la excepción de las señales begintransfer y beginbursttransfer. Un esclavo del tipo Avalon-MM puede activar la señal waitrequest durante los ciclos libres. Un maestro del tipo Avalon-MM puede iniciar una transferencia cuando la señal waitrequest esté activada y espera a que la señal se desactive.
Señales Pipeline			
readdatavalid *readdatavalid_n*	1	Esclavo → Maestro	Usada para transferencias de lectura del tipo pipeline y de latencia variable. Activada por el esclavo para indicar que la señal readdata contiene datos validos en respuesta a una petición de lectura mediante la activación de la señal read. Un esclavo que incorpore la señal readdatavalid debe activar esta señal durante un ciclo por cada acceso de lectura

			recibido. Debe existir al menos un ciclo de latencia entre la recepción de la señal read correspondiente y la activación de la señal readdatavalid. Esta señal es necesaria si el maestro soporta lecturas del tipo pipelined. También debe de incorporarse en maestros tipo bursting con funcionalidad de lectura incorporada.
Señales Burst			
burstcount	1-11	Maestro → Esclavo	Utilizada por maestros de tipo burst para indicar el número de transferencias en cada burst. La sincronización de la señal burstcount se controla mediante la propiedad constantBurst. Los maestros del tipo burst con la funcionalidad de lectura deben incluir la señal readdatavalid.
beginbursttransfer	1	Maestro → Esclavo	Activada durante el primer ciclo de un burst para indicar cuando comienza una transferencia del tipo burst. Esta señal se desactiva después de un ciclo independientemente del valor de la señal waitrequest.

2.2.2.2 Propiedades de la interfaz

Nombre	**Valor por defecto**	**Valores permitidos**	**descripción**
addressUnit	Maestro-symbol Esclavo-word	Word, symbol	Especifica la unidad de los datos para lecturas y escrituras.
burstCountBoundariesOnly	Words	True, false	Si es true, se garantiza que todas las transferencias tipo burst presentes en la interfaz comienzan en una dirección que es múltiplo del tamaño de burst en bytes.
constantBurstBehavior	Maestro-true Esclavo-false	True, false	Cuando su valor es true para un maestro, declara que el maestro mantiene la señal address y la de burstcount durante el burst; cuando su valor es falso, declara que el maestro mantiene la señal address y burstcount sólo para la primera transacción del burst. Cuando su valor es true para un esclavo, declara que el esclavo espera una

			señal address y burstcount estable durante todo el burst; cuando su valor es false, declara que el esclavo toma sólo el valor de la señal de address y de burstcount durante la primera transacción del burst.
holdTime (1)	0	0-1000 cycles	Especifica el tiempo en timingUnits entre la activación de la señal write y la desactivación de las señales chipselect, address y data. (Sólo se aplica a transferencias de tipo escritura)
linewrapBurst	False	True, false	Algunos dispositivos de memoria implementan un sistema de wrapping burst en vez de un burst tipo incremental. La diferencia entre estos dos tipos es que un burst tipo wreapping, cuando capta la dirección de un burst boundary, la dirección se ajusta al límite previo del burst de tal manera que sólo los bit de orden bajo se requieren para la identificación de la dirección.
maximumPendingReadTransactions (1)	1 (2)	1-64	El número máximo de lecturas pendientes que pueden ser puestas en cola por el esclavo.
readLatency (1)	0	0-63	Fija la latencia en las transferencias de lectura para esclavos Avalon-MM. No se utiliza en interfaces que incluyan la señal readdatavalid.
readWaitTime (1)	1	0-1000 cycles	Para interfaces que no usan la señal waitrequest, readWaitTime indica el número de ciclos o nanosegundos antes que el esclavo acepte una orden de lectura. El tiempo es como si el esclavo activara la señal waitrequest durante los ciclos del readwaittime.
setupTime (1)	0	0-1000 cycles	Especifica el tiempo en timingUnits entre la activación de las señales chipselect, address y data y la activación de las señales read o write.
timingUnits (1)	Cycles	Cycles, nanoseconds	Especifica las unidades para setupTime, holdTime, writeWaitTime y readWaitTime. Usa los ciclos para dispositivos síncronos y los nanosegundos para dispositivos asíncronos.
writeWaitTime (1)	0	0-1000 cycles	Para interfaces que no usan la señal waitrequest, writeWaitTime indica el número de ciclos o nanosegundos antes que un esclavo acepte una transferencia de escritura. El tiempo es como si el esclavo

			activara la señal waitrequest durante los ciclos o nanosegundos indicados por el valor de writeWaitTime.
Propiedades de relación en la interfaz			
associatedClock	---	---	Nombre de la interfaz del reloj para sincronizar la interfaz Avalon-MM.
associatedReset	---	---	Nombre de la interfaz de reset para sincronizar la interfaz Avalon-MM.
bridgeToMaster	Null	Maestro Avalon-MM en el mismo componente	Un bridge Avalon-MM consiste en un esclavo y un maestro, y tiene la propiedad de permitir el acceso ante una petición del esclavo a un byte o bytes concretos y que causará que el maestro realice la petición para el mismo byte o bytes. El componente Avalon-MM Pipelined Bridge en la librería de componentes de Qsys implementa esta funcionalidad.

(1) Aunque esta propiedad caracteriza un dispositivo esclavo, los maestros pueden declarar esta propiedad para permitir conexiones directas entre interfaces de unión para esclavos y maestros.

(2) Si un componente acepta más transferencias de lectura que el valor que aquí se indica, la FIFO interna de lectura pendiente puede sobrecargarse con datos erróneos, incluyendo la perdida de la señal readdata, conexión a la interfaz de un maestro equivocado de la esta misma señal o un apagado del sistema

2.2.2.3 Transferencias

- Conceptos básicos:

  - Una transferencia es una operación de escritura y lectura de una palabra, un símbolo o un dato, entre un puerto del tipo Avalon-MM y la interconexión. El sistema Avalon-MM transfiere palabras en un rango de tamaño desde 8 hasta 1024 bits. Las transferencias tardan una o más ciclos de reloj para completarse.
  - Pareja maestro-esclavo es un término que hace referencia a los puertos de maestro y esclavo que se ven involucrados durante una transferencia. Durante una transferencia, el control del puerto maestro y las señales de datos pasan a través de las interconexiones de fábrica e interactúan con el puerto esclavo.

- Transferencias típicas de lectura y escritura:

  Un esclavo puede mantener la interconexión tantos ciclos como sean necesarios para activar la señal de waitrequest. Si un esclavo usa dicha señal para una transferencia de lectura o escritura, entonces debe usar la señal waitrequest para ambas.

  Si un esclavo recibe las señales de address, byteenable o write, y la señal writedata después de un flanco de subida del reloj, el puerto esclavo debe activar la señal waitrequest antes del siguiente flanco de subida para mantener las transferencias. Cuando un esclavo activa la señal waitrequest, la transferencia es retrasada y las señales de dirección y control se mantiene constantes durante dicho periodo. Las transferencias se completan durante el primer flanco de subida de la señal de reloj (clk) después de que el puerto esclavo desactive la señal de waitrequest.

  No existe un límite de tiempo para que el puerto esclavo pueda mantener dicho estado. Por lo tanto es importante asegurarse que la señal waitrequest no permanece activada de manera indefinida.

  La señal de waitrequest puede desacoplarse de las señales read y write de manera que puede ser activada durante los ciclos desocupados. Un maestro del tipo Avalon-MM puede iniciar una transacción cuando la señal waitrequest se activa y esperar a que dicha señal sea desactivada. Desacoplando la señal waitrequest de las señales read y write puede mejorar la sincronización del sistema gracias a la eliminación de bucles combinacionales incluidos en las señales de read, write y waitrequest.

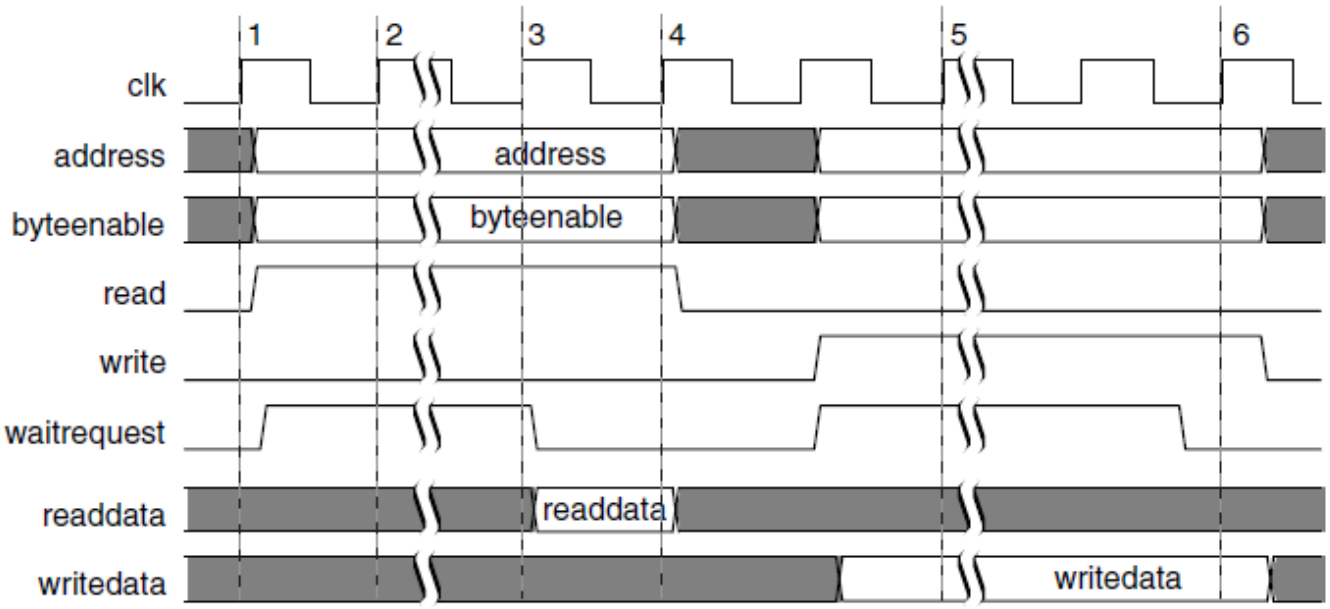

Figura 5: Transferencia de lectura y escritura con la señal waitrequest [5]

(1) Las señales addres, read y begintrasfer se activan después del flanco de subida de la señal clk, la señal waitrequest se activa manteniendo así la transferencia.
(2) La señal waitrequest es recogida. Debido a esto, el ciclo se transforma en un ciclo de estado de espera, y las señales address, read, write y byteenable permanecen constantes. La señal begintransfer no se mantiene constante.
(3) El esclavo presenta una señal valida de readdata y desactiva la señal waitrequest.
(4) La señal readdata se descativa y la señal waitrequest es recogida, completando la transferencia.
(5) Las señales address, writedata, byteenable, begintransfer, y write se activan. El esclavo responde activando la señal waitrequest, manteniendo la transferencia.
(6) El esclavo captura la señal writedata y desactiva la señal de waitrequest, finalizando la transferencia.

❖ Transferencias de lectura y escritura con estados de espera prefijados:

En vez de usar la señal waitrequest para mantener una transferencia, un esclavo puede especificar estados de espera fijos usando las propiedades readWaitTime y WriteWaitTime. Las señales de control y direcciones (byteenable, read y write) se mantienen constantes durante la duración de la transferencia. Las sincronizaciones de lectura/escritura con las propiedades readWaitTime/writeWaitTime establecido a un valor <n> es exactamente el mismo que cuando se activa la señal waitrequest durante <n> ciclos por transferencias.

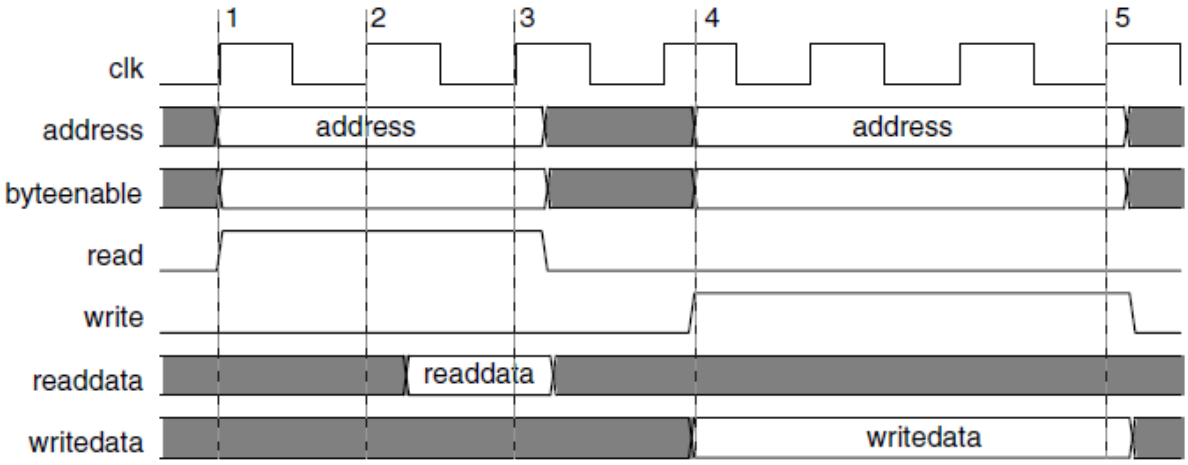

Figura 6: Transferencias de lectura y escritura con estados fijos de espera en una interfaz esclava [5]

(1) El maestro activa la señal address y read durante el flanco de subida del clk.
(2) El siguiente flanco de subida del clk marca el final del primer y único ciclo de estado de espera ya que el valor de la propiedad readWaitTime es 1.
(3) El esclavo captura la señal readdata durante el flanco de subida del clk, y la transferencia de lectura termina.
(4) Las señales writedata, address, byteenable y write están disponibles para ser capturadas por el esclavo.
(5) Debido a que el valor de la propiedad writeWaitTime es 2, la transferencia finaliza después de completarse. Las señales de datos y de control se mantienen constantes hasta este momento.

Este tipo de transferencias se suelen utilizar para periféricos externos multiciclos. Los periféricos pueden capturas las señales de control y de direcciones durante el flanco de subida del clk, y tiene un ciclo completo para devolver los datos. Los componentes sin ningún estado de espera están permitidos, pero pueden disminuir el valor de frecuencias alcanzables por culpa de la generación de la respuesta en el mismo ciclo que la petición.

❖ Transferencias del tipo Pipeline:

Las transferencias de lectura del tipo Avalon-MM pipelined incrementan el rendimiento para dispositivos esclavos síncronos que requieren varios ciclos para devolver los datos durante un primer acceso, pero pueden devolver un valor de dato por ciclo durante un tiempo tras éste. Nuevas transferencias de lectura del tipo pipelined pueden comenzar antes que se devuelva la señal de readdata de otras previas. Las transferencias de escritura no pueden ser del tipo pipelined.

Una transferencia del tipo pipelined se divide en dos fases: una fase de direcciones y otra de datos. Un maestro inicia una transferencia mediante la presencia de un valor de dirección durante la primera fase; un puerto esclavo completa la transferencia mediante la entrega de datos durante la fase de datos. La fase de direcciones de una nueva transferencias (o transferencias múltiples) pueden comenzar antes de que se complete la transferencia de una fase de datos previa. El retraso se define como latencia de pipeline, que es la duración desde el final de la fase de direcciones hasta el comienzo de la fase de datos.

Las diferencias claves en el efecto en la sincronización de la transferencia entre los estados de espera y la latencia de pipeline son las siguientes:

- Los estados de espera determinan la duración de la fase de direcciones, y el límite del rendimiento máximo de un puerto. Si un esclavo requiere un estado de espera para responder a una petición de transferencia, entonces el puerto necesita al menos dos ciclos de reloj por transferencia.
- La latencia de pipeline determina el tiempo hasta que se devuelven los datos independientemente de la fase de direcciones. Un puerto esclavo del tipo pipelined sin estado de espera puede realizar una transferencia por ciclo, aunque esto puede necesitar algunos ciclos de latencia para devolver las primeras unidades de datos.

Los estados de espera y las lecturas del tipo pipelined pueden ser implementados de manera concurrente, y la latencia de pipeline puede ser fija o variable.

❖ Transferencias de lectura de tipo pipeline con latencia variable:

Un esclavo Avalon-MM de tipo pipeline necesita un ciclo o más para producir los datos después de que las señales de dirección y control sean capturadas. Un puerto esclavo del tipo pipeline puede presentar múltiples transferencias de lectura pendientes en cualquier momento. Una transferencia de lectura del tipo pipeline con latencia variable usa el mismo conjunto de señales que una transferencia de lectura normal, añadiendo la señal adicional de readdatavalid. Los periféricos esclavos que usan la señal readdatavalid son considerados como del tipo pipeline con latencia variable; las señales de readdata y readdatavalid pueden activarse el ciclo después de que el ciclo de lectura este activado.

El puerto esclavo debe devolver el valor de la señal readdata al mismo tiempo que se aceptan las direcciones. Los puertos esclavos del tipo pipeline con latencia variable deben usar la señal waitrequest. Los esclavos pueden activar la señal waitrequest para parar la transferencia manteniendo así el número de transferencias pendiente en un nivel aceptable.

El número máximo de transferencias pendientes es una propiedad de la interfaz del esclavo. La interconexión de fábrica construye la lógica que direcciona la señal de readdata a los maestros que lo solicitan.

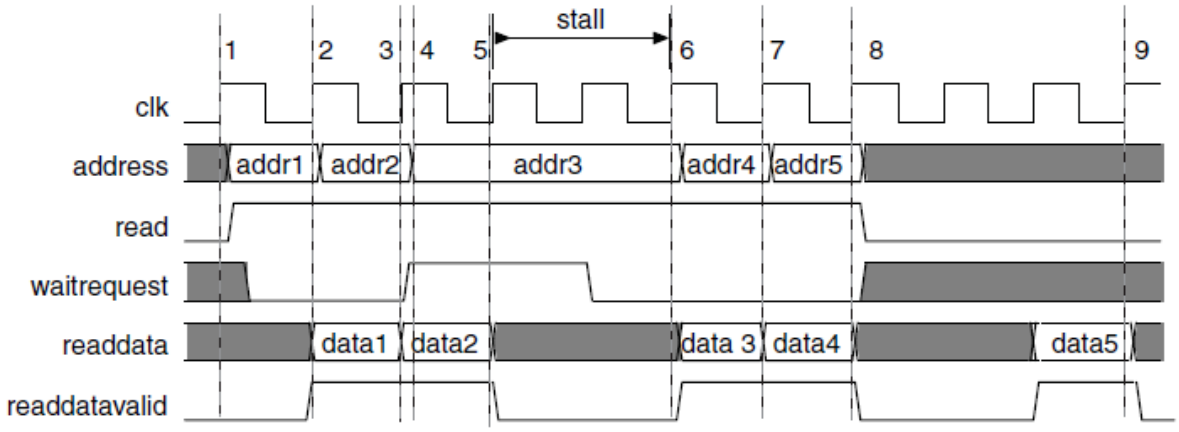

Figura 7: Transferencia de lectura del tipo pipeline con latencia variable [5]

(1) El maestro activa las señales de address y read, iniciando una transferencia de lectura.
(2) El esclavo captura el valor addr1, e inmediatamente proporciona la respuesta data1 y activa la señal readdatavalid.
(3) El esclavo captura el valor addr2 e inmediatamente proporciona la respuesta data2 y activa la señal readdatavalid. La interconexión captura el valor data1.
(4) El esclavo activa la señal waitrequest durante dos ciclos causando que la tercera transferencia permanezca en espera.
(5) La interconexión captura el valor data2.
(6) El esclavo direcciona la señal readdatavalid y un valor valido de la señal readdata en respuesta a la tercera transferencia de lectura.
(7) El dato de la transferencia 3 es capturado por la interconexión al mismo tiempo que se captura el valor addr4 por parte del esclavo.
(8) El esclavo captura el valor addr5. La interconexión captura el valor data4.
(9) Se recibe el valor data5 junto la señal readdatavalid completando así la fase de datos para la transferencia pendiente final de lectura.

Si el esclavo no puede manejar una transferencia de escritura mientras está procesando una transferencia pendiente de lectura, el esclavo debe activar la señal waitrequest y parar las operaciones de escritura hasta que se completen las transferencias de lectura pendientes. Las especificaciones de la interfaz Avalon-MM no define el valor de la señal readdata en el caso que un esclavo acepte una transferencia de escritura para la misma dirección mientras una transferencia de lectura permanezca pendiente. Los esclavos del tipo pipeline con latencia variable deben de presentar la señal waitrequest.

❖ Transferencia de lectura del tipo pipeline con latencia fija:

La fase de direcciones para transferencias de lectura con latencia fija es idéntica al caso de una de latencia variable. Después de la fase de direcciones, un puerto esclavo del tipo pipeline con latencia fija necesita un número fijo de ciclos de reloj para devolver un señal válida de readdata, tal y como se indica en la propiedad readWaitTime. La interconexión captura la señal readdata en el flanco de subida apropiado, y la fase de datos termina.

Durante la fase de direcciones, el puerto esclavo puede activar la señal waitrequest para mantener la transferencia o puede especificar un valor de la propiedad readWaitTime para un número fijo de estados de espera. La fase de direcciones termina en el siguiente flanco de subida del reloj después de los estados de espera, en caso de que los haya.

Durante la fase de datos, el esclavo direcciona la señal readdata después una latencia fija. Si el esclavo tiene una latencia de lectura de valor <n>, el puerto esclavo debe presentar una señal de readdata valido en el flanco de subida del reloj número <n> después del fin de la fase de direcciones.

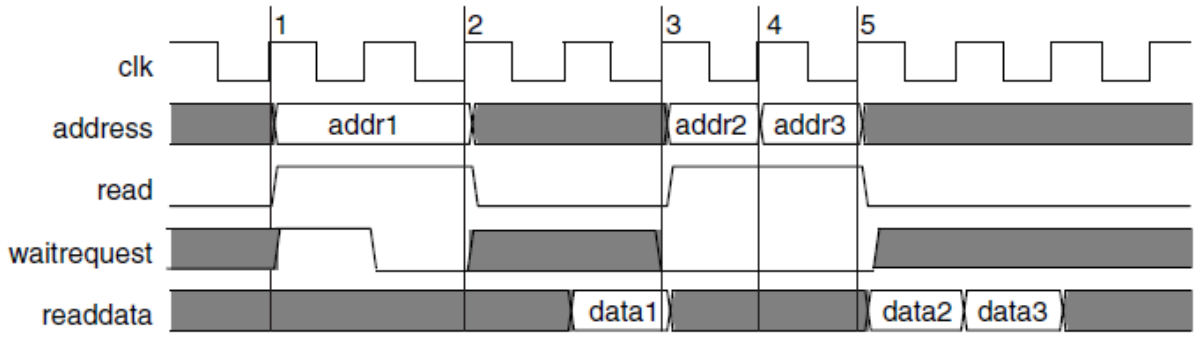

Figura 8: Transferencia de lectura del tipo pipeline con latencia fija de dos ciclos [5]

(1) Un maestro inicia una transferencia de lectura mediante la activación de la señal read y el valor addr1. El esclavo activa la señal waitrequest para mantener la transferencia durante un ciclo.
(2) El esclavo desactiva la señal waitrequest y captura el valor addr1 durante el flanco de subida del reloj. La fase de direcciones termina aquí.
(3) El esclavo presenta un valor válido de readdata después de dos ciclos, terminando así la transferencia.
(4) El valor addr2 y la señal read se activan para una nueva transferencia de lectura.
(5) El maestro inicia una tercera transferencia de lectura durante el siguiente ciclo, antes que el dato de la transferencia previa sea devuelto.

❖ Transferencias tipo burst:

Un burst ejecuta transferencias múltiples como si fueran una única unidad, en lugar de tratar a cada palabra de manera independiente. Los bursts puede incrementar el rendimiento para puertos esclavos que consiguen una eficiencia mayor cuando manejan múltiples palabras al mismo tiempo. El efecto de realizar un burst es bloquear el arbitraje durante su duración. Si una interfaz Avalon-MM incluye funcionalidades de lectura y escritura, y a su vez permite la realización de transferencias tipo burst, debe permitir tanto lecturas como escrituras de este tipo.

Para soportar bursts, una interfaz Avalon-MM incluye una señal de salida burstcount. Si un esclavo tiene un burstcount input, se considera capacitado para soportar este tipo de transferencias.

La señal de burstcount se comporta de la siguiente manera:

- Al comienzo de un burst, la señal burstcount presenta el número de transferencias secuenciales in el burst.
- Para una anchura <n> de la señal burstcount, la longitud máxima de burst es $2^{(<n>-1)}$. La mínima longitud permitida para el burst es 1.

Para soportar burst de lectura esclavos, un esclavo debe también permitir:

- Estados de espera con la señal waitrequest.
- Transferencias del tipo pipeline con latencia variable mediante la señal readdatavalid.

Al comienzo de un burst, el esclavo comprueba la señal address y el valor de longitud del burst reflejado por la señal burstcount. Durante un burst con una dirección de valor <a> y un valor de la señal burstcount de <b>, el esclavo debe ejecutar <b> transferencias consecutivas comenzando en la dirección <a>. El burst se completa después de que el esclavo reciba (señal write) o bien devuelva (señal de read) la palabra de datos número <b>. EL esclavo de tipo burst debe capturar la señal address y la de burstcount sólo una vez por cada burst. La lógica del esclavo debe inferir la dirección para todas ellas excepto la primera transferencia en el burst. Un esclavo puede también usar la señal de input beginbursttransfer, que es activada por la interconexión durante el primer ciclo de cada burst.

❖ Write burst:

Cuando comienza un burst de escritura con una señal de burstcount mayor que una, se aplican las siguientes reglas:

- Cuando llega un burstcount de valor <n> al comienzo del burst, el esclavo debe aceptar <n> unidades sucesivas de valores de la señal writedata para completar el burst. El arbitraje entre la pareja maestra-esclava se bloquea hasta que el burst se completa, garantizando así la llegada de los datos desde el puerto maestro que inició el burst.
- El esclavo debe sólo capturar la señal writedata cuando se activa la señal write. Durante el burst, la señal write puede ser desactivada para indicar que no existe un valor válido de la señal writedata. Desactivar la señal write no finaliza el burst; sólo lo retrasa. Cuando un burst se retrasa, ningún otro maestro puede acceder al esclavo, reduciendo la eficiencia de la transferencia.
- La propiedad constantBurstBehavior controla la conducta de las señales del burst. Cuando su valor es true para un maestro, declara que el maestro mantiene la señal address y burstcount a lo largo del burst; cuando su valor es false, declara que el maestro mantienela señal address y burstcount sólo durante la primera transacción del burst. Cuando su valor es true para un esclavo, declara que el esclavo espera recibir una señal address y burstcount estable a lo largo de todo el burst; cuando es false, declara que el esclavo recoge la señal address y burstcount sólo durante la primera transacción de un burst.
- El esclavo puede retrasar una transferencia mediante la activación de la señal waitrequest que fuerza mantener constante el valor de las señales writedata,write y byteenable.
- La funcionalidad de la señal byteenable es la misma tanto para esclavos de tipo burst y los normales. La señal byteenable puede cambiar de valor para palabras diferentes durante el burst.

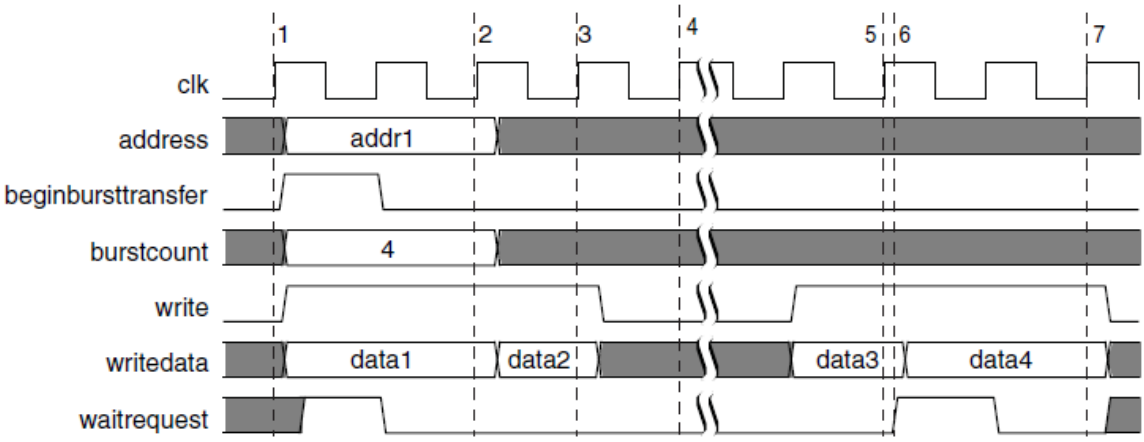

Figura 9: Burst de escritura con la propiedad constantBurstBehavior establecida en un valor de false tanto para el maestro como para el esclavo [5]

(1) El maestro activa las señales address, burstcount, write y direcciona el primer valor de writedata. El esclavo inmediatamente activa la señal waitrequest, indicando que todavía no está preparado para comenzar con la transferencia.
(2) La señal waitrequest se encuentra a nivel bajo, el esclavo captura los valores addr1, burstcount, y el primer valor de la señal writedata. En los siguientes ciclos de la transferencia las señales address y burstcount son ignoradas.
(3) El puerto esclavo captura la segunda unidad de datos durante el flanco de subida del reloj.
(4) El burst se detiene mientras la señal write esté desactivada.
(5) El esclavo captura la tercera unidad de datos durante el flanco de subida del reloj.
(6) El esclavo activa la señal waitrequest. En respuesta, todas las salidas se mantienen constantes durante otro ciclo de reloj.
(7) El esclavo captura la última unidad de datos durante el flanco de subida del reloj. El burst de escritura finaliza.

❖ Burst de lectura:

Los bursts de lectura son similares a las transferencias de lectura de tipo pipeline con latencia variable. Un burst de lectura tiene fases de direcciones y de datos diferentes, y la señal readdatavalid indica cuando el esclavo presenta un valor válido de la señal readdata. La diferencia es que un único burst de lectura de dirección proporciona transferencias de datos múltiples.

Las reglas que se aplican a los burst de lectura son las siguientes:

- Cuando el valor de la señal burstcount es <n>, el esclavo devuelve <n> palabras de la señal readdata para completar el burst.
- El esclavo proporciona cada palabra de la señal readdata mediante la activación de señal readdatavalid durante un ciclo. La desactivación de readdatavalid retrasa el burst pero no finaliza la fase de datos.
- La señal byteenable presente en un burst de lectura ordena las solicitudes para todos los ciclos del burst. Un valor de byteenbale de 1 significa que el bit menos significante es el que se lee durante todos los ciclos de lectura.

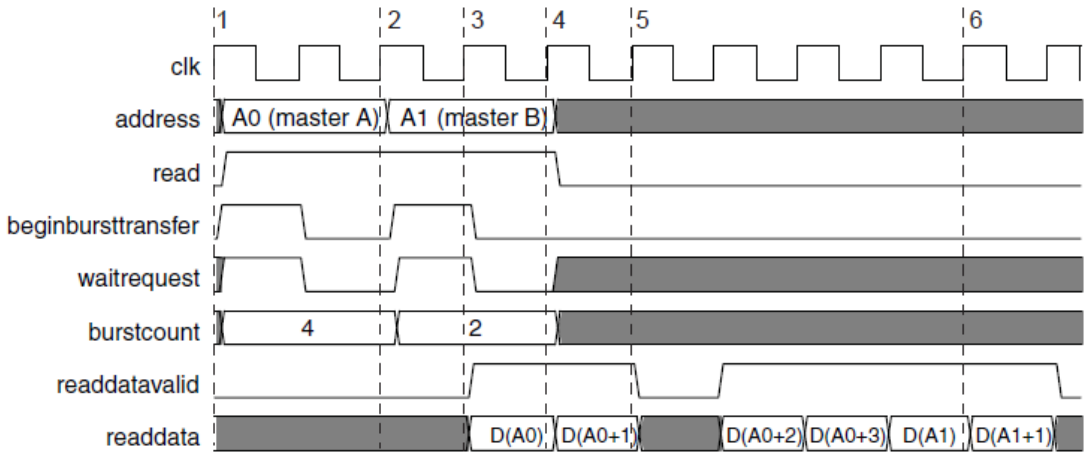

Figura 10: Burst de lectura [5]

(1) El maestro A activa la señal address(A0), burstcount y read después del flanco de subida del reloj. El esclavo activa la señal waitrequest, causando que todos los inputs excepto la señal beginbursttransfer se mantengan constantes durante otro ciclo de reloj.
(2) El esclavo captura el valor A0 y burstcount durante el flanco de subida de reloj. Una nueva transferencia podría comenzar durante el siguiente ciclo.
(3) El maestro B direcciona la señal address (A1), burstcount y read. El esclavo activa la señal waitrequest, causando que todos los inputs excepto la señal beginbursttransfer se mantengan constantes. El esclavo podría haber devuelto datos de lectura desde la primera solicitud de lectura.
(4) El esclavo presenta un valor de señal readdata válido y activa la señal readdatavalid, transfiriendo la primera palabra de datos del maestro A.
(5) La segunda palabra del maestro A es transferida. El esclavo desactiva la señal readdatavalid pausando el burst de lectura. El puerto esclavo puede mantener la señal readdatavalid desactivada durante un número indefinido de ciclos de reloj.
(6) Se devuelve la primera palabra del maestro B.

❖ Burst de tipo Line-Wrapped:

Procesadores con caches de datos o instrucciones consiguen eficiencia mediante el uso de burst tipo line-wrapped. Cuando un procesador solicita datos, y el dato no se encuentra en la caché, el controlador de la caché recoge suficientes datos desde la memoria para rellenar la línea entera de caché.

❖ Alineación de las direcciones:

Para sistemas en los que la anchura de datos entre el maestro y el esclavo sean diferentes, la interconexión realiza las tareas de alineación entre las direcciones. La interfaz Avalon-MM resuelve las diferencias entre la anchura de los datos, de manera que cualquier puerto maestro pueda comunicarse con cualquier puerto esclavo, independientemente su anchura de datos respectivos. La interconexión sólo soporta accesos alineados; un maestro sólo puede emitir direcciones que sean un múltiplo de su anchura de datos. (Un maestro puede escribir palabras parciales mediante la desactivación de algunos de los bits de la señal byteenable).

2.2.3 Otras interfaces

❖ Interfaces Avalon de interrupción:

Las interfaces Avalon de interrupción permiten a los componentes esclavos señalizar eventos a los componentes maestros.

➢ **Interrupt Sender**

Un interrupt sender direcciona una única señal de interrupción a un interrupt receiver. La sincronización de la señal irq debe ser síncrona respecto al flanco de subida de su reloj asociado, pero no guarda relación con cualquier otra transferencia o interfaz. La señal irq debe estar activada hasta que la interrupción sea recogida en la interfaz Avalon-MM esclava.

➢ **Interrupt Receiver**

Una interfaz del tipo interfaz recibe las interrupciones enviadas por las interfaces de interrupt sender. Los componentes con una interfaz maestra Avalon-MM puede incluir un interrupt receiver para detectar la activación de interrupciones por parte de los componentes esclavos con interfaces del tipo interrupt sender. Los interrupt receiver aceptan las peticiones de interrupción de cada interrupt sender como bits separados.

❖ Interfaz Avalon Streaming (Avalon-ST):

Este tipo de interfaz permite la conexión directa entre una fuente (source) y un sumidero (sink), creando un flujo de datos unidireccional permitiendo la transferencia de alta velocidad de datos ya que elimina el arbitraje intermedio del sistema.

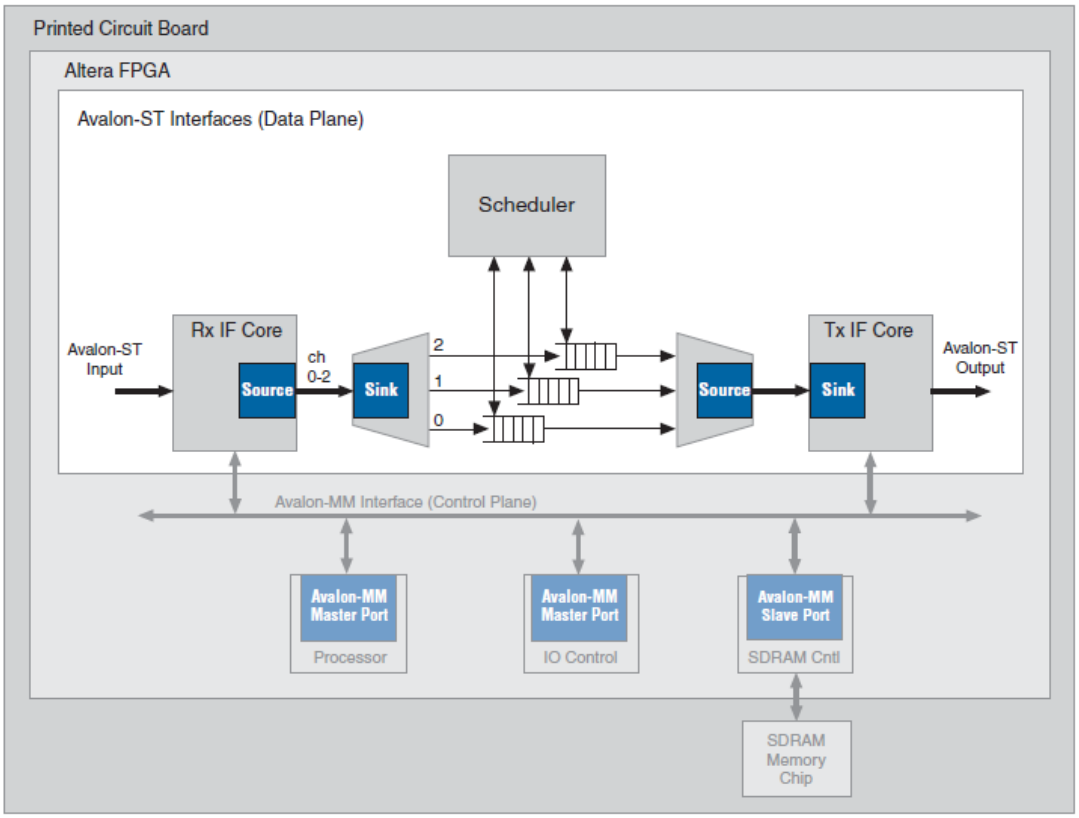

Figura 11: Ejemplo típico de aplicación de una interfaz Avalon-ST [5]

Entre las principales características de este tipo de interfaz:

- Baja latencia, alto rendimiento de transferencias de datos punto a punto.
- Soporte de múltiple canal con paquetes flexibles entrelazados.
- Señalización de banda lateral de canal, error y comienzo y final de delineación de paquete.
- Soporte para bursting de datos.
- Adaptación automática de la interfaz.

❖ Interfaz Avalon Conduit:

Las interfaces de tipo Avalon Conduit se utilizan para colecciones arbitrarias de señales para ser exportadas al siguiente nivel de un diseño jerárquico o al top level de un sistema Qsys. Una interfaz Avalon Conduit puede incluir input, output y señales bidireccionales. Direcciones, tales como las fuentes y los sumideros de las interfaces Avalon-ST, o las entradas y salidas de maestros y esclavos de una interfaz Avalon-MM, no se aplican a las interfaces de tipo Avalon Conduit. Un módulo puede tener múltiples interfaces Avalon Conduit para proporcionar agrupaciones lógicas de las señales que van a ser exportadas.

❖ Interfaz Avalon Tristate Conduit:

La interfaz Avalon Tristate Conduit (Avalon-TC) es una interfaz punto a punto diseñada para controladores on-chip que direccionan a componentes externos. Esta interfaz permite a los pines de datgos, direcciones y control ser compartidos por múltiples dispositivos de tipo triestado.

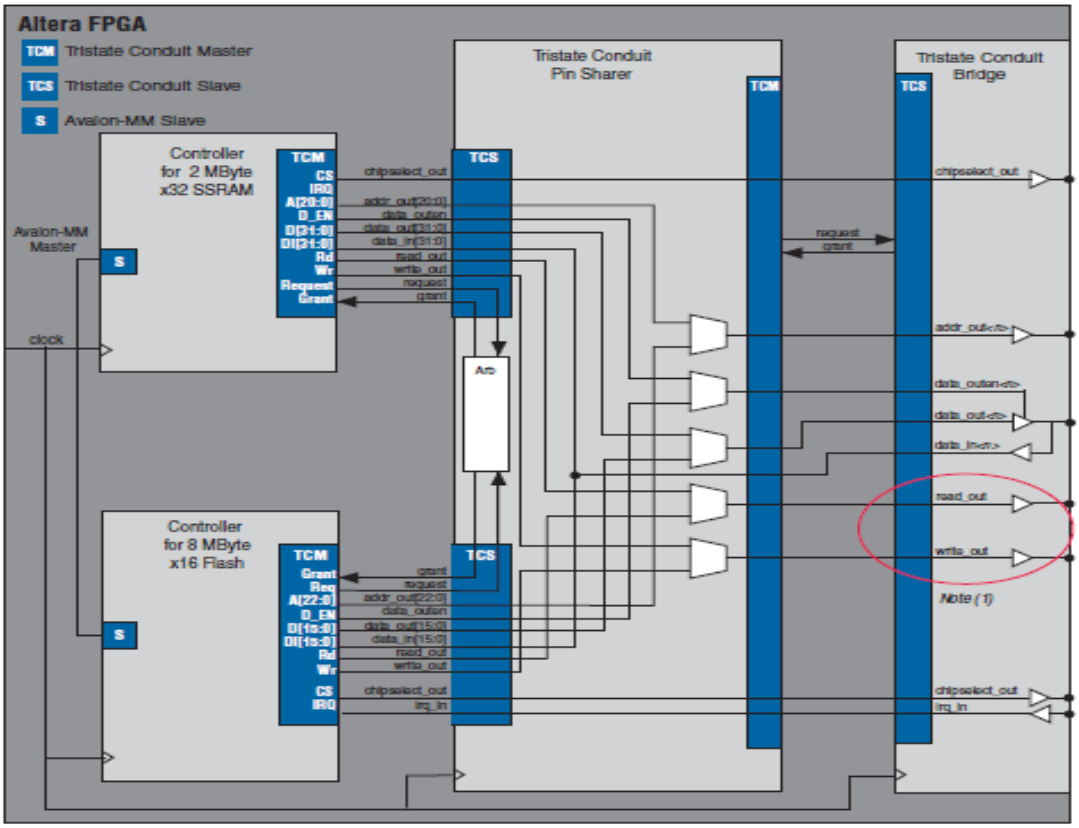

Figura 12: Ejemplo de interfaz Avalon Tristate Conduit [5]

3. Adaptación del módulo de detección de fallos

3.1 Descripción general del diseño original y su funcionalidad.

Tal y como se refleja en el diseño original [9], el proceso completo de funcionamiento del módulo se compone de tres fases diferentes:

- Configuración previa a la síntesis, en donde se adaptan los parámetros internos al sistema Nios II en el que va a ser incorporado..

- Programación del módulo ya sintetizado para la supervisión de bloques de comunicación (*rutinas*). Este procedimiento consiste fundamentalmente en indicar al módulo de detección la información necesaria para realizar la supervisión de cada una de las rutinas seleccionadas.

- Finalmente sólo queda controlar las dos ejecuciones de la rutina necesarias para realizar cada una de las comprobaciones de funcionamiento correcto.

En la Figura 13 se se puede visualizar de forma gráfica el funcionamiento del módulo de detección para el tratamiento de una rutina. Una vez implementado el sistema, se envía al módulo la configuración de la rutina. Permitiendo así proceder a la supervisión de la misma durante su funcionamiento.

La configuración de rutinas consiste en indicar al módulo de detección las direcciones de inicio y de fin de la rutina, así como el número máximo de ciclos de reloj en el que debe completarse la ejecución de dicha rutina. El formato de dirección necesario para pasar esta información al módulo de detección se describe en el apartado 3.3.1.

Terminada la configuración del módulo, se pasa a la fase de vigilancia del bus de comunicaciones esperando que se produzca una transferencia del maestro al esclavo deseado y en una de las direcciones previamente almacenadas como direcciones de inicio. Es entonces cuando se inicia la supervisión de una de las rutinas y se genera una firma. Simultáneamente se activa un temporizador *watchdog* que activa la señal de error de ejecución en caso de que se sobrepase el número máximo de ciclos de reloj indicado en el banco de registros.

Por otro lado, si el maestro realizase una escritura durante la dirección de fin de la rutina en curso antes de que desborde el *watchdog*, se considera que dicha rutina ha finalizado. Al finalizar la rutina se coge la firma resultante almacenada en el banco de registros si se trata de la primera ejecución de la rutina, o bien, en caso de ser la segunda ejecución, se compara con la firma resultante de la primera rutina. En este último caso, se activaría la señal de error de ejecución si las firmas son distintas entre sí.

Finalizado este proceso el módulo estaría de nuevo preparado para entrar en una nueva rutina.

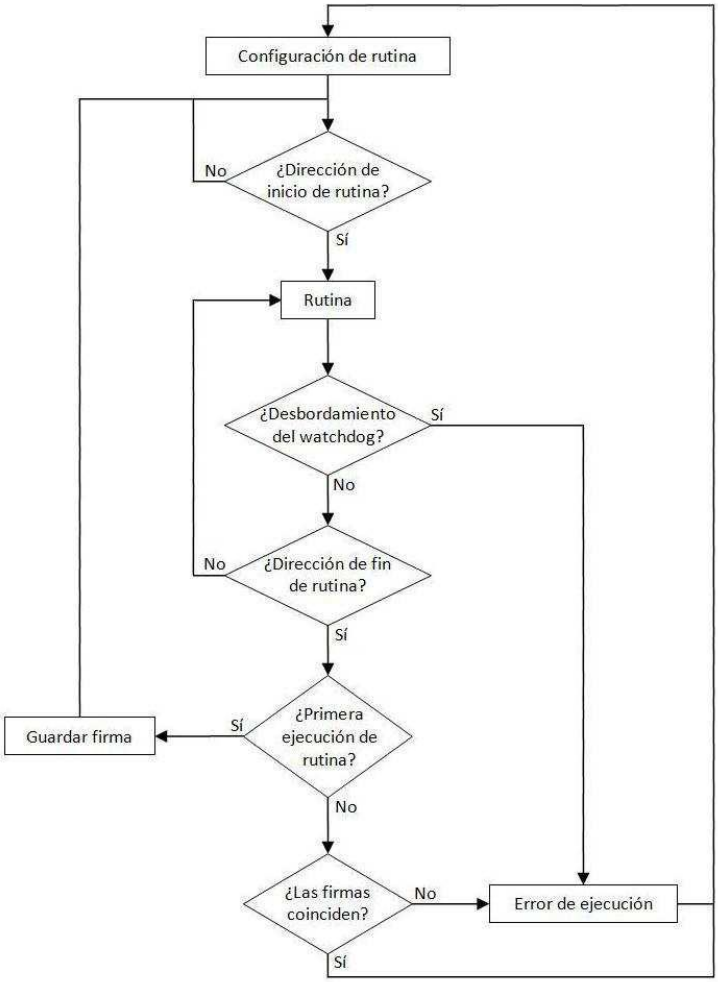

Figura 13: Flujograma de operación del módulo [9]

El diseño se divide en cuatro sub-módulos, tal y como muestra la Figura 14:

- *Interfaz*, que es la interfaz de comunicaciones con el bus Avalon.
- *Regbank*, banco de registros que almacena la configuración de cada una de las rutinas así como el resultado de la primera ejecución de cada rutina.
- *Control*, máquina de estados que determina el funcionamiento del módulo.
- *Espía*, bloque encargado de generar la firma a partir del bus de datos del bus Avalon.

El funcionamiento detallado de cada uno de estos bloques se expone a continuación.

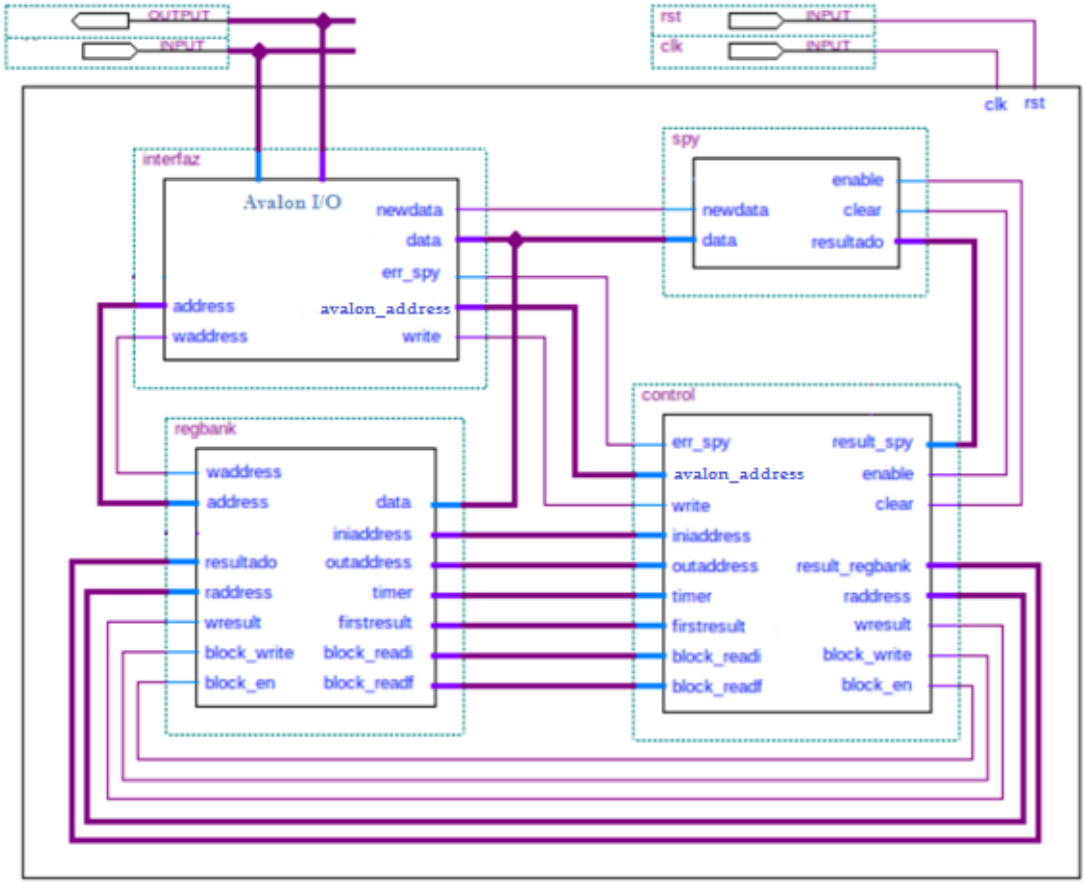

Figura 14: Diagrama de bloques del módulo

3.2 Estrategias de adaptación del módulo de detección original

Durante el desarrollo del presente proyecto se adoptaron dos estrategias diferentes para conseguir incorporar el módulo original a un sistema Avalon.

La primera de ellas fue la incorporación al sistema Nios II de un bridge que hiciese la labor de traducción del conjunto de señales salientes del maestro Avalon, en las señales necesarias correspondientes de un esclavo AMBA.

Para ello se utilizó un bridge específico [8] que se incorporó como esclavo mediante la herramienta Qsys. Dicho bridge exportaba las señales del bus de datos Avalon a un nivel superior de sistema, lo que permitía la conexión de elementos lógicos externos.

Una vez realizada la configuración del sistema se procedió a realizar una serie de pruebas al conjunto llegándose a las siguientes conclusiones:

- Al tener que instanciar el sistema Nios II dentro de un proyecto general desarrollado en el software Quartus II, las labores de simulación y síntesis se complicaban considerablemente. Este nuevo inconveniente se debía fundamentalmente a la imposibilidad de utilizar las propias herramientas facilitadas por Altera para la generación de los testbench necesarios.

- No se generaban todas las señales necesarias para el funcionamiento del módulo de detección.

- Debido al hecho de situar un bridge entre el módulo de detección y el propio bus de datos, todas las señales de entrada sufrían el retraso de un ciclo de reloj hasta ser recibidas por el módulo. Este hecho generaba un problema a la hora de una vigilancia óptima del sistema, por lo que se optó finalmente por buscar una nueva alternativa de adaptación.

La segunda estrategia que se consideró, fue la posibilidad de adaptar directamente el propio módulo al nuevo protocolo de comunicaciones Avalon. Esta estrategia solventaba completamente los dos inconvenientes de la anterior, ya que permitía la integración directa del módulo dentro del sistema Nios II. Pudiéndose por tanto utilizar directamente las herramientas desarrolladas por Altera para la simulación y síntesis del sistema, y eliminando el problema del retraso de las señales al existir un bridge.

Tras un estudio del módulo original se comprobó que para adaptarlo a un sistema Nios II, únicamente era necesario rediseñar el bloque Interfaz y eliminar la interrupción por error de escritura.

En el punto siguiente se describen las modificaciones que fueron necesarias para su incorporación al diseño final.

3.3 Adaptación del módulo de detección original

En este capítulo se describen todos los aspectos relevantes que han involucrado la adaptación del diseño original del módulo de detección de fallos; describiendo sus señales, funcionamiento y configuración, así como las modificaciones realizadas para adaptarlo al nuevo protocolo de comunicaciones Avalon.

La finalidad del módulo de detección es la supervisión de las comunicaciones a través del bus principal del sistema para detectar los errores transitorios que se produzcan [9]. Como ya se mencionó previamente, el módulo original desarrollado utilizaba una interfaz adaptada al bus AMBA, esto supone la necesidad de realizar diversas modificaciones en las que se sustituirán las señales del bus antiguo por las del bus Avalon.

El módulo de detección se ha adaptado para funcionar como un esclavo del bus Avalon cuya misión es supervisar las comunicaciones de un maestro a uno o varios esclavos específicos. Aunque con algunas modificaciones mínimas del diseño, tanto en el módulo de detección como en su posterior configuración en la herramienta Qsys, se podría emplear para espiar las comunicaciones en sentido contrario añadiendo las señales de lectura correspondientes, (avs_s0_read y avs_s0_readdata).

En la adaptación del diseño del módulo de detección se ha considerado que no se efectúan transferencias de tipo burst (Modo ráfaga) [5], en caso de ser así, se debería adecuar el diseño del sistema de detección a las particularidades de este tipo de transferencia teniendo en cuenta las características propias definidas en el apartado de la interfaz Avalon.

En los siguientes apartados se describe en profundidad la adaptación del diseño original del módulo de detección, mientras que en el capítulo siguiente se explica la inserción del módulo completo dentro del sistema Nios II.

3.3.1 Bloque Interfaz

La Figura 15 muestra las señales de entrada y salida del bloque de interfaz adaptado, las cuales se detallan a continuación.

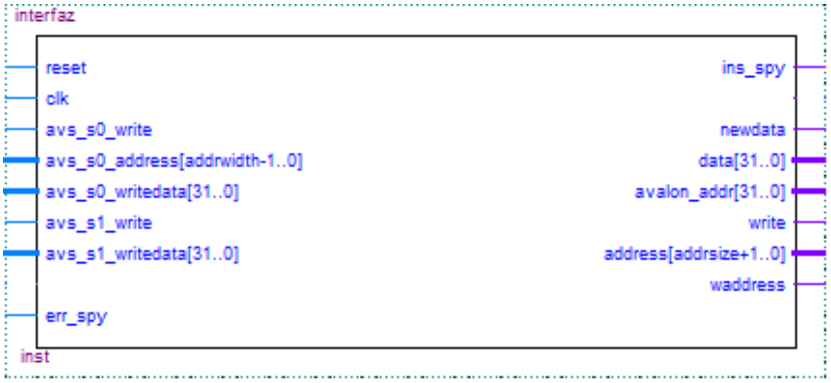

Figura 15: Bloque de interfaz

Señales de entrada:

- *Reset*: Reset asíncrono, activo por nivel bajo.
- *Clk*: Señal de reloj.
- *Avs_s0_write*: Entrada de habilitación de escritura al esclavo objetivo.
- *Avs_s1_write*: Entrada de habilitación de escritura al módulo de detección.

- *Avs_s0_address*: Entrada de direcciones del bus Avalon.
- *Avs_s0_writedata*: Entrada de datos de escritura del bus Avalon.
- *Avs_s1_writedata*: Entrada de datos de escritura del bus Avalon, (no tiene utilidad en el diseño, se incorpora para cumplir las especificaciones del bus Avalon-MM).
- *Err_spy*: Indica que se ha detectado un error de ejecución.

Señales de salida:

- *Newdata*: Indica que hay un nuevo dato en el bus Avalon.
- *Data*: Bus de datos del bus Avalon.
- *Avalon_addr*: Bus de direcciones del bus Avalon.
- *Write*: Indica que se produce una transferencia del maestro al esclavo seleccionado.
- *Address*: Dirección para escritura de los registros de configuración del banco de registros.
- *Waddress*: Habilitar escritura de dirección en el banco de registros.

En el proceso de adaptación del diseño original [9], el bloque interfaz ha sido el que más cambios ha sufrido de cara a su incorporación en un sistema basado en el microprocesador Nios II. En primer lugar ha sido necesario eliminar todas las señales propias correspondientes al protocolo de comunicaciones AMBA. Este conjunto de señales se englobaban en los grupos ahbsi y ahbso.

Por razones de arquitectura a la hora de implementar el módulo en el sistema final utilizando la herramienta Qsys, también se ha optado por eliminar la señal de interrupción por error de escritura en el módulo (err_w), aunque la funcionalidad de protección de escritura sigue vigente y su aplicación sigue siendo la misma con la excepción de generar la interrupción correspondiente.

Dentro del conjunto de señales genéricas también se ha procedido a la eliminación de las siguientes; hindex, venid, devid, versión, nummaster y numslave. Dicha decisión se toma ya que este conjunto de señales se incluían por obligatoriedad a la hora de implementar el diseño original en un sistema basado en el protocolo AMBA y por tanto ya no son necesarias.

Respecto a las señales que comunican la interfaz con el resto de bloques del módulo, todas han permanecido sin alterar a excepción de la ya mencionada err_w.

En cuanto a su funcionalidad, el bloque interfaz es el encargado de gestionar las comunicaciones con el bus Avalon. El funcionamiento de la interfaz modificada se describe de forma gráfica en la Figura 16, con excepción de la generación de interrupciones. A grandes rasgos, la interfaz vigila constantemente el bus de comunicaciones a la espera de una transferencia entre el maestro objetivo y el esclavo objetivo, o bien una comunicación con el propio módulo de detección de fallos. Estas transferencias vienen determinadas por la activación de la señal avs_s0_write, en el caso de comunicación con el esclavo objetivo, o la señal avs_s1_write en el caso de comunicación con el módulo de detección de errores.

Tal y como se indica en [9], si se ejecuta una transferencia del maestro objetivo al esclavo objetivo, la interfaz señala al resto de bloques del módulo que se ha producido una transferencia susceptible de ser vigilada. En ese caso, se activa la señal *write* en la fase de dirección de dicha transferencia, indicando al bloque de control que se trata de una operación de escritura entre el maestro y el esclavo deseados, y la señal *newdata* durante el primer ciclo de reloj de la fase de datos de esa transferencia, indicando al bloque espía que hay un nuevo dato presente en el bus para que lo capture.

Por otra parte, si se produce una comunicación con el propio módulo de detección y la transferencia es correcta, se habilita la escritura en el banco de registros. Para que una dirección se considere válida debe encontrarse dentro del rango adecuado definido por el sistema configurado en Qsys. Algunas consideraciones a tener en cuenta son:

- Los dos bits menos significativos del bus de direcciones del bus Avalon se ignoran, ya que el Nios II es un microprocesador de 32 bits y por tanto realiza los incrementos de dirección de 4 en 4.

- Dependiendo de la anchura de direcciones asociadas a cada periférico esclavo se deberá comprobar el número de bits necesarios para determinar los valores a configurar en la señal de direccionamiento del banco de registros (*address*), para cumplir los requisitos necesarios de funcionamiento, la señal de direcciones asociada al periférico debe tener un mínimo de cinco bits.

- Debido al procedimiento interno de traducción de direcciones realizado por el sistema Qsys, el valor de salida de direcciones desde el microprocesador Nios II y el de llegada a cada uno de los periféricos es diferentes. Este valor se adapta en función del rango de direcciones asociado a cada uno de ellos y por tanto es necesario reconfigurar el sistema del módulo de detección para cada uno de los diseños realizados mediante el parámetro addrwidth.
- Dentro de la señal avs_s0_address se utilizará el valor genérico addrsize para definir los bits a utilizar (addrsize+2 bits) para direccionar la señal address. Dentro de este grupo de bits, los dos más significativos indican el tipo de registro al que se accede, (dirección de inicio de rutina, dirección final de rutina, o número máximo de ciclos permitidos para ejecutarse la rutina correspondiente), mientras que el resto representan el índice de rutina.

Figura 16: Flujograma de operación de la interfaz

Por otro lado, la interfaz también se encarga de generar la interrupción asociada a los errores de ejecución cuando lo indica el banco de registros y el bloque de control por medio de la señal *err_spy*.

Aunque en principio el bloque descrito anteriormente se adapta a un gran número de diseños de sistemas Nios II, es importante reseñar que se pueden encontrar aplicaciones específicas en las que las transferencias realizadas entre maestro-esclavo no sean las típicas. En ese caso, es posible que sea necesario incorporar otras señales definidas en el apartado 2.2.2.1, tales como waitrequest, o personalizar alguna de las propiedades. Este ajuste permitiría conseguir que el módulo de detección fuese capaz de captar de manera óptima los valores de las señales correctos para cada una de las rutinas a ser vigiladas.

Finalmente es importante recordar que las conexiones realizadas por la herramienta Qsys son automáticas, no permitiendo, por ejemplo, direccionar la señal avs_s0_write para detectar las transferencias de escritura entre el maestro y el esclavo objetivos. Por ello, será necesario modificar a posteriori los archivos Verilog HDL sintetizados correspondientes al sistema Nios II, para sustituir dicha señal por la que sea requerida para el correcto funcionamiento de nuestro módulo.

3.3.2 Bloque Banco de registros

La Figura 17 muestra la interfaz del banco de registros. A continuación se detalla la función de cada una de las señales de las que consta.

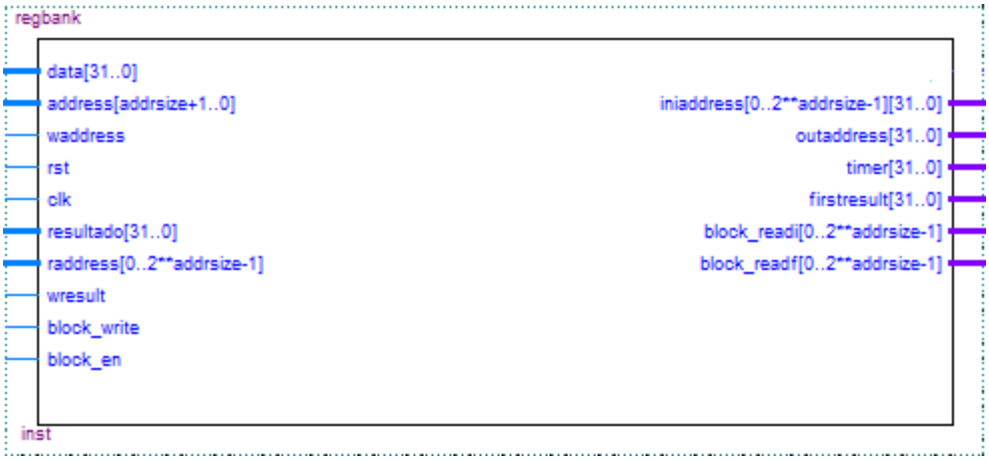

Figura 17: Banco de registros

Señales de entrada:

- *Rst*: Reset asíncrono, activo por nivel bajo.
- *Clk*: Señal de reloj.
- *Data*: Bus de datos del bus Avalon.
- *Address*: Dirección para escritura de los registros de configuración del banco de registros.
- *Waddress*: Habilitar escritura de dirección en el banco de registros.
- *Resultado*: Firma resultante de la primera ejecución de una rutina para almacenar en registro.
- *Raddress*: Dirección de los registros de resultado del banco de registros, y dirección para lectura de los registros de configuración. Se corresponde con el índice de la rutina en ejecución.
- *Wresult*: Habilitar escritura de resultado en el banco de registros.
- *Block_write*: Modificación de la protección contra escritura.
- *Block_en*: Habilitar modificación de la protección contra escritura.

Señales de salida:

- *Iniaddress*: Direcciones de inicio de todas las rutinas.
- *Outaddress*: Dirección de fin de la rutina actual.
- *Timer*: Número máximo de ciclos de reloj permitidos para la rutina actual.
- *Firstresult*: Resultado de la primera ejecución de la rutina actual.
- *Block_readi*: Estado de la protección contra lectura de las direcciones de inicio.
- *Block_readf*: Estado de la protección contra lectura de las direcciones de fin.

Respecto al diseño original, este bloque sólo ha sufrido una pequeña modificación; la eliminación de la señal err_w. El resto de funcionalidades definidas en el bloque inicial han permanecido inalteradas ejecutándose de manera análoga a las descritas en el apartado 3.1.3 de [9].

3.3.3 Paquete genérico mytypes

Se trata de un archivo creado en sustitución del bloque Genéricos original para definir señales personalizadas utilizadas en el propio módulo de detección. En este caso, únicamente se utilizará una del tipo array, (banco_reg), para almacenar los valores de dirección de inicio de rutina en el registro (reginicio), así como algunas de las señales internas utilizadas por algunos de los bloques del módulo.

3.3.4 Resto de bloques

El resto de los bloques originales diseñados para el módulo de detección; control y spy, no sufre ninguna modificación. Pueden verse sus cararacterísiticas en los apartados 3.1.4 y 3.1.5 de [9].

4. Configuración del sistema Nios II

En este capítulo se especifican las características de diseño del sistema basado en el procesador Nios II. Para su configuración se utilizará la herramienta Qsys integrada en Quartus II.

En este capítulo se describirá la construcción de un sistema cuyo objetivo es ejecutar un programa de contador binario para el encendido secuencial de 8 LEDS, este sistema fue el utilizado para la verificación inicial del funcionamiento correcto del módulo IP pero su procedimiento es extensivo a cualquier otro diseño que se quiera realizar en la herramienta Qsys.

4.1 Características

El sistema descrito a continuación es un pequeño sistema basado en el procesador Nios II que mediante un periférico de entrada salida, permite el encendido y apagado de LEDs siguiendo el patrón de un contador binario. Este sistema también permite la conexión con una computadora permitiendo a esta misma controlar la lógica dentro de la FPGA.

El sistema Nios II contiene los siguientes componentes:

- El núcleo del procesador Nios II.
- Una memoria interna.
- Un módulo de temporización.
- Un módulo JTAG UART
- Un puerto I/O de 8 bits en paralelo para controlar los LEDs.
- El módulo de detección de errores diseñado.

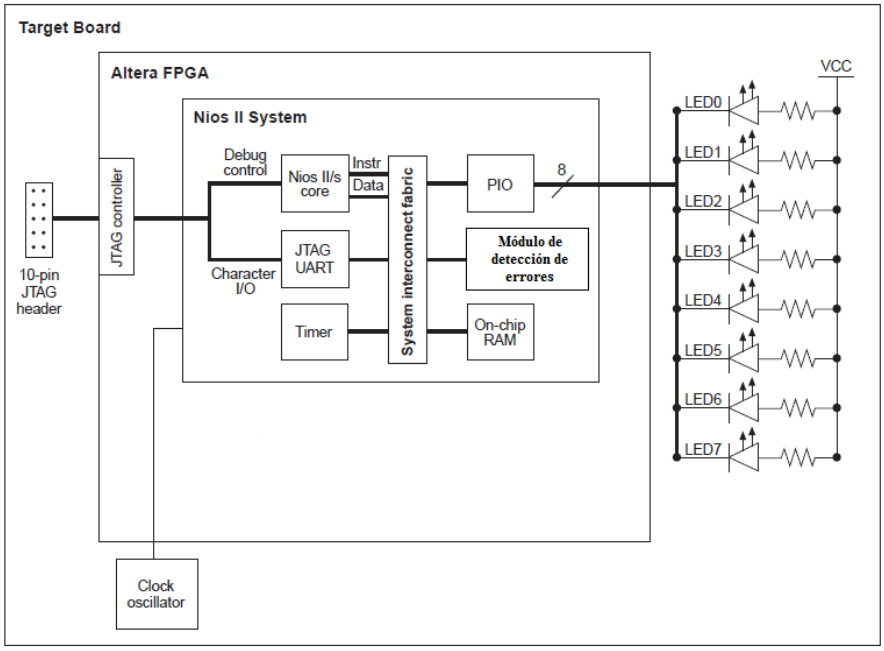

Figura 18: Diagrama de bloques del sistema Nios II diseñado

4.2 Módulos incorporados

❖ Memoria interna:

Cualquier sistema basado en un microprocesador necesita al menos una memoria para datos e instrucciones. En este diseño se utilizará una memoria interna de tipo RAM de 20 KB para ambos.

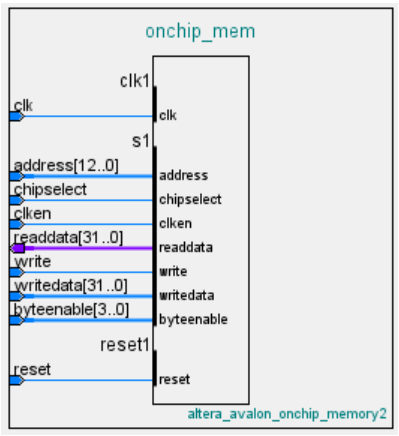

Figura 19: Módulo de memoria utilizado en el diseño

Como se puede ver en la figura nº 19, el módulo de memoria tiene un total de ocho señales de entrada y una de salida. Es importante resaltar que todas estas señales se corresponden a las ya definidas anteriormente en la interfaz Avalon.

Todas estas señales y las de los módulos expuestos más adelante, se conectan de manera automática mediante la propia herramienta Qsys a las señales correspondientes del núcleo del procesador Nios II. Únicamente es necesario definir físicamente en la herramienta cual es la conexión entre el esclavo y el maestro, de manera que una vez confirmada es el propio software de configuración el que se encarga conectar las señales correspondientes pertenecientes a cada una de las interfaces entre sí.

❖ Módulo de temporización:

La mayoría de los sistemas controlados utilizan un componente de temporización para permitir una sincronización correcta y adecuada de todo el diseño. En este caso se utilizará el componente Interval Timer, perteneciente a la librería.

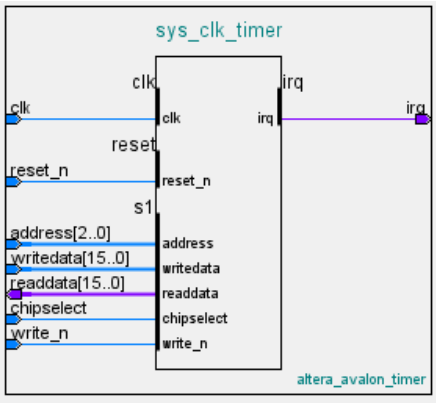

Figura 20: Módulo de temporización utilizado en el diseño

Como se puede ver en la figura nº 20, el módulo de temporización tiene un total de seis señales de entrada y dos de salida. Es importante resaltar nuevamente que todas estas señales se corresponden a las ya definidas anteriormente en el estándar de la interfaz Avalon.

❖ Módulo de JTAG UART:

Este módulo permite una manera adecuada de comunicar datos con el procesador Nios II mediante un cable de descarga USB-Blaster. Entre su principal funcionalidad es la descarga del software diseñado para el dispositivo programable y su configuración previa al uso.

En este caso seleccionaremos el módulo prediseñado correspondiente dentro de la librería de la herramienta Qsys.

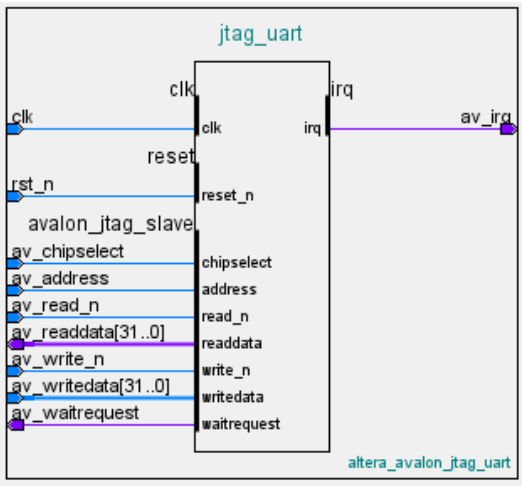

Figura 21: Módulo JTAG UART utilizado en el diseño

Como se puede ver en la figura nº 21, el módulo JTAG UART tiene un total de siete señales de entrada y tres de salida. Es importante resaltar nuevamente que todas estas señales se corresponden a las ya definidas anteriormente en el estándar de la interfaz Avalon.

❖ Puerto I/O de 8 bits en paralelo para controlar los LEDs:

Las señales controladas por un PIO proporcionan a los sistemas basados en el procesador Nios II una manera sencilla de recibir señales de input y de direccionar señales de output. Se pueden implementar tantos como sean necesarios en función de la complejidad del sistema diseñado, viéndose únicamente limitado por las capacidades propias de la FPGA en la que se empotrará el sistema.

En este diseño se utilizará el módulo PIO prediseñado dentro de la librería Qsys, con una configuración de output de 8 bits que se activarán siguiendo una secuencia de contador binario.

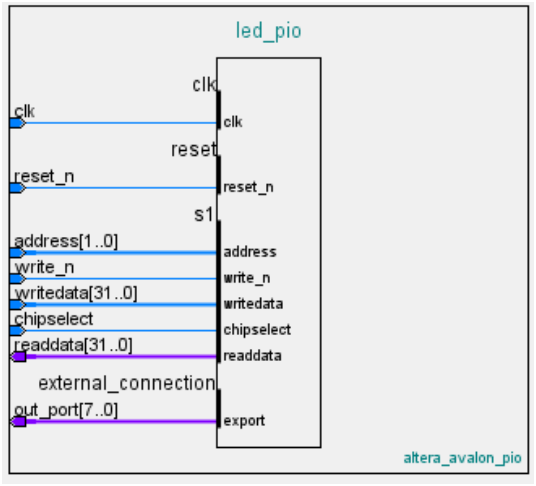

Figura 22: Módulo PIO utilizado en el diseño

Como se puede ver en la figura nº 22, el módulo PIO tiene un total de seis señales de entrada y dos de salida, una de las cuales (out_port[7..0]) se exportará fuera del propio sistema diseñado por la herramienta Qsys. Es importante resaltar nuevamente que todas estas señales se corresponden a las ya definidas anteriormente en el estándar de la interfaz Avalon.

❖ Módulo del núcleo del procesador Nios II:

Para el diseño seleccionado se utilizará un núcleo del tipo Nios II/s core. Esta configuración está especialmente diseñada para sistemas que requieren un tamaño menor de núcleo. Entre las principales características de esta configuración tipo se pueden resaltar:

- Tiene una caché de instrucciones, pero no de datos.
- Puede permitir hasta un espacio de direcciones externas de 2 GB.
- Emplea un pipeline de cinco estados.
- Permite la predicción de saltos estáticos.
- Proporciona la posibilidad de multiplicar, dividir o cambiar el hardware para mejorar el rendimiento aritmético.
- Permite la incorporación de instrucciones personalizadas.
- Proporciona la posibilidad de instalar un módulo de JTAG debug.

En el diseño del sistema a utilizar, se configuró una memoria caché de instrucciones de 2 KB. Tampoco se utilizó la opción de multiplicación o división del hardware, ni se habilitaron las transferencias tipo burst.

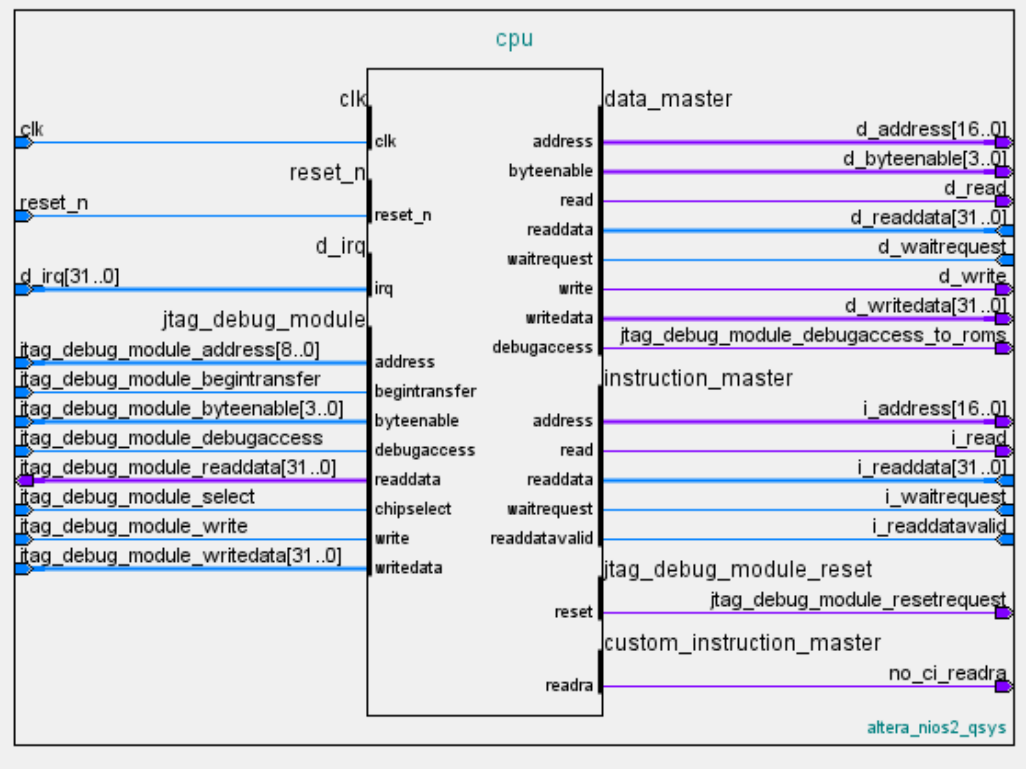

Figura 23: Módulo del Nios II /s core utilizado en el diseño

Como puede verse en la figura 23, las señales de entrada y salida del core se pueden agrupar en cinco grupos diferentes.

En primer lugar tenemos las señales generales de funcionamiento del sistema. En este grupo encontramos la señal del reloj y de reset, así como la entrada de las señales de interrupciones (irq).

En segundo lugar encontramos el bloque de señales correspondiente al sistema JTAG debug que se mencionó previamente en las características de este núcleo. En el caso particular de nuestro diseño, estas señales no se utilizan en ningún momento.

El tercer grupo de señales se corresponden con el maestro de datos del núcleo. En el encontramos un total de seis outputs y dos inputs que proporcionan todas las señales necesarias para el correcto funcionamiento del sistema bajo las interfaces ya descritas del bus Avalon. En el caso del actual diseño, con estas señales se trabajará en el módulo de detección de errores y por tanto presentan una gran importancia su tratamiento y conexión dentro del sistema.

El cuarto grupo de señales está conformado con el maestro de instrucciones del core. Presenta un total de dos outputs y tres inputs. Este grupo de señales se encarga de manejar el flujo de ejecución del programa mediante la configuración designada en el software cargado. En este caso, únicamente se conectará al módulo de memoria del sistema.

El último grupo, está compuesto por dos señales específicas que, en el caso particular del diseño utilizado, no tienen funcionalidad alguna, ya que ni se utilizará el JTAG debug ni se crearán instrucciones personalizadas.

❖ Módulo de detección de errores:

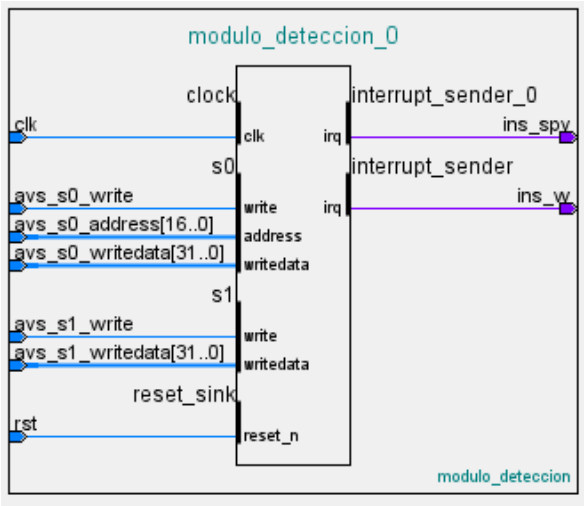

Figura 24: Módulo de detección de errores utilizado en el diseño

Este módulo, descrito en el capítulo anterior, se ha diseñado utilizando lenguaje VHDL, partiendo de otro diseñado específicamente para realizar la misma tarea sobre un sistema LEON 3. Para su incorporación en la herramienta Qsys, se utilizó la funcionalidad de añadir nuevo componente. Para ello se necesitó definir las interfaces adecuadas de conexión, y el rol de todas las señales contenidas, figura 25.

Name	Interface	Signal Type	Width	Direction
clk	clock	clk	1	input
avs_s0_write	s0	write	1	input
avs_s0_address	s0	address	addrwidth	input
avs_s0_writedata	s0	writedata	32	input
avs_s1_write	s1	write	1	input
avs_s1_writedata	s1	writedata	32	input
ins_spy	interrupt_sender_0	irq	1	output
rst	reset_sink	reset_n	1	input

Figura 25: Asignación de interfaces y tipos de señal del módulo de detección

La descripción en profundidad del diseño y la funcionalidad de este módulo se encuentra en el capítulo anterior.

4.3 Configuración de interrupciones y del rango de direcciones

Como puede en la figura 26, la herramienta Qsys nos permite distribuir los rangos de direcciones adjudicados a cada uno de los esclavos de manera personalizada.

	cpu.data_master	cpu.instruction_master
onchip_mem.s1	0x00088000 - 0x0008cfff	0x00088000 - 0x0008cfff
cpu.jtag_debug_module	0x00090800 - 0x00090fff	0x00090800 - 0x00090fff
jtag_uart.avalon_jtag_slave	0x00091030 - 0x00091037	
sys_clk_timer.s1	0x00091000 - 0x0009101f	
led_pio.s1	0x00091020 - 0x0009102f	
modulo_deteccion_0.s0	0x00000000 - 0x0007ffff	
modulo_deteccion_0.s1	0x00091038 - 0x0009103b	

Figura 26: Mapa de direcciones configurable del sistema Nios II

Esto permite ajustar los valores de dirección que utilizarán todos nuestros periféricos ofreciendo así la posibilidad de reconfigurar la lógica del módulo de detección para simplificar su funcionamiento para cada diseño

Respecto a los valores de las interrupciones externas utilizadas en el diseño, la herramienta Qsys proporciona la posibilidad de adjudicarle, desde la propia pestaña de "system contents" un valor entero del 0 al 31 a cada una de ellas. Esta numeración permite establecer un orden de prioridad a cada una de ellas en función del valor adjudicado. Ver figura 27.

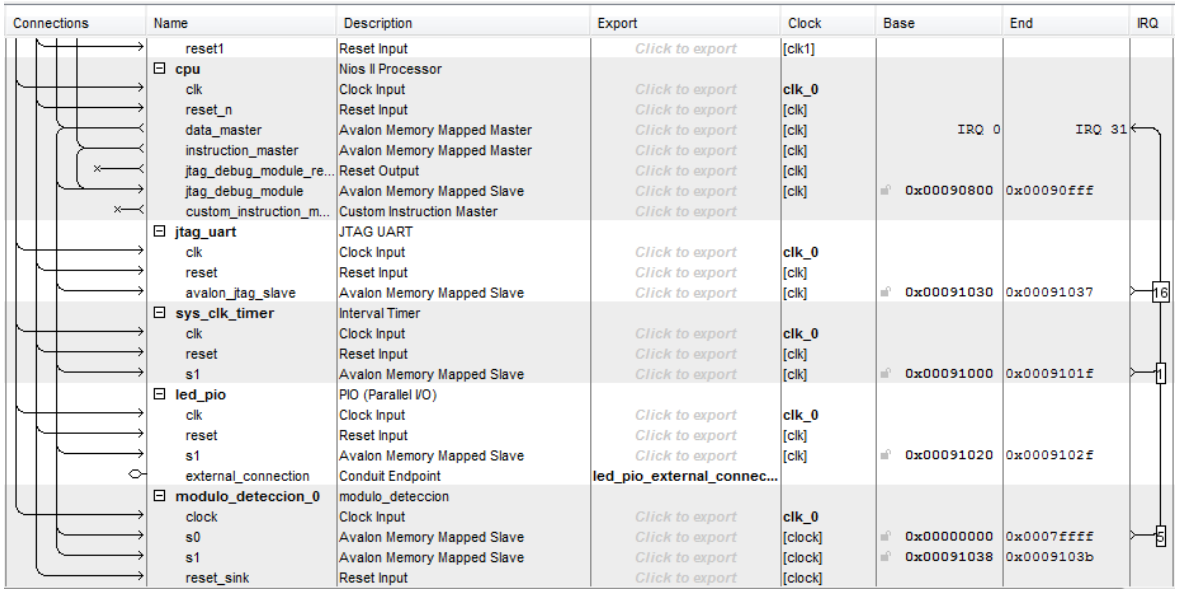

Figura 27: Pestaña de system contents de la herramienta Qsys

5. Resultados

Una vez concluido el diseño del módulo de detección de fallos se ha procedido a realizar un conjunto de pruebas sobre el mismo, que pueden clasificarse en tres grupos:

- *Validación de funcionalidad*, consistente en la simulación del módulo de detección de fallos, tanto por separado como conectado a un sistema embebido, para verificar su funcionalidad. Se corresponde con la fase de depuración del diseño.

- *Síntesis del diseño*, en este caso se determinarán los requisitos de espacio y frecuencia de funcionamiento del módulo de detección en relación al microprocesador Nios II.

- *Test de errores*, donde se comprueba la capacidad del módulo de detección de fallos mediante la inserción de fallos en la simulación del propio módulo conectado a un sistema basado en el microprocesador Nios II.

A continuación se explica en profundidad cada una de estas pruebas y se muestran los resultados obtenidos.

5.1 Validación de la funcionalidad del diseño

La finalidad del este conjunto de pruebas es comprobar que la funcionalidad del diseño se corresponde con lo esperado, todas estas simulaciones se realizaron de manera iterativa para depurar el diseño hasta la obtención de una versión óptima cuyo resultado sea el esperado previamente. Para realizar la simulación de los diseños realizados se ha empleado el simulador Modelsim.

La depuración del diseño se ha dividido en tres fases. Por un lado se ha probado la funcionalidad del módulo de detección sin la interfaz, es decir, completamente aislado. Por otro se ha verificado el funcionamiento de la interfaz, por separado, conectada al bus Avalon. Por último, se ha realizado una comprobación del módulo de detección al completo conectado al bus Avalon. En este apartado se muestran únicamente los resultados de la última versión del módulo de detección, plenamente funcional.

5.1.1 Simulación del módulo de detección sin interfaz

En esta primera fase de simulación se comprueba la funcionalidad del diseño sin conectar al bus Avalon.

Para poder realizar el procedimiento de manera adecuada, ha sido necesaria la modificación de la entidad de más alto nivel del diseño, (Spylvl), para excluir el bloque Interfaz. Esto implica, que las señales de comunicación de la interfaz con el resto de bloques del módulo de detección, pasan a ser los puertos de entrada y salida de la nueva entidad de más alto nivel.

Para todas las pruebas englobadas en esta validación, se ha considerado un valor de addrsize de 3, es decir, un total máximo de 8 rutinas soportadas.

Como es habitual la simulación se realiza mediante la preparación de un banco de pruebas programado en VHDL utilizando la herramienta Quartus II.

Se han realizado tres simulaciones del diseño para esta fase:

- Figuras 28 a 30: funcionamiento normal sin errores de ejecución.
- Figuras 31 y 32: funcionamiento normal con errores de ejecución y escritura.
- Figura 33: prueba de protección contra escritura.

Todas las versiones comienzan con un reset del módulo de detección y la posterior configuración de las rutinas. Al activar a nivel bajo el reset, se borran todos los datos almacenados en el banco de registros, (reginicio, regfin, regtime y regresult) y se activan las protecciones contra lectura de todas las rutinas a vigilar, (block_ri y blck_rf). Además, se desactivan las protecciones contra escritura para las rutinas (block_w) y lleva al estado de espera al bloque de control, asignando un 0 lógico al resto de registros del módulo, (check,watchdog, pos y result).

A continuación se configura el módulo de detección para observar dos rutinas diferentes asignadas a las posiciones 1 y 5 del banco de registros. La primera de las rutinas comienza en la dirección hexadecimal 00CA30 y termina en 00CA44, mientras que las direcciones de inicio y final para la segunda de las rutinas son, respectivamente, 00CA50 y 00CA5F. A continuación se establecen los límites máximos del tiempo de ejecución para cada una de ellas, 40 ciclos para la primera y 30 para la segunda.

En la figura 28 puede verse la inicialización de funcionamiento del módulo, así como la configuración de las rutinas a ser vigiladas. Cuando se activa la señal waddress, se actualiza el valor del registro correspondiente con el valor presente en ese momento en el bus de datos (data). También se puede comprobar como cada vez que se realiza dicha operación, se desactiva la protección contra lectura asignada a dicho registro.

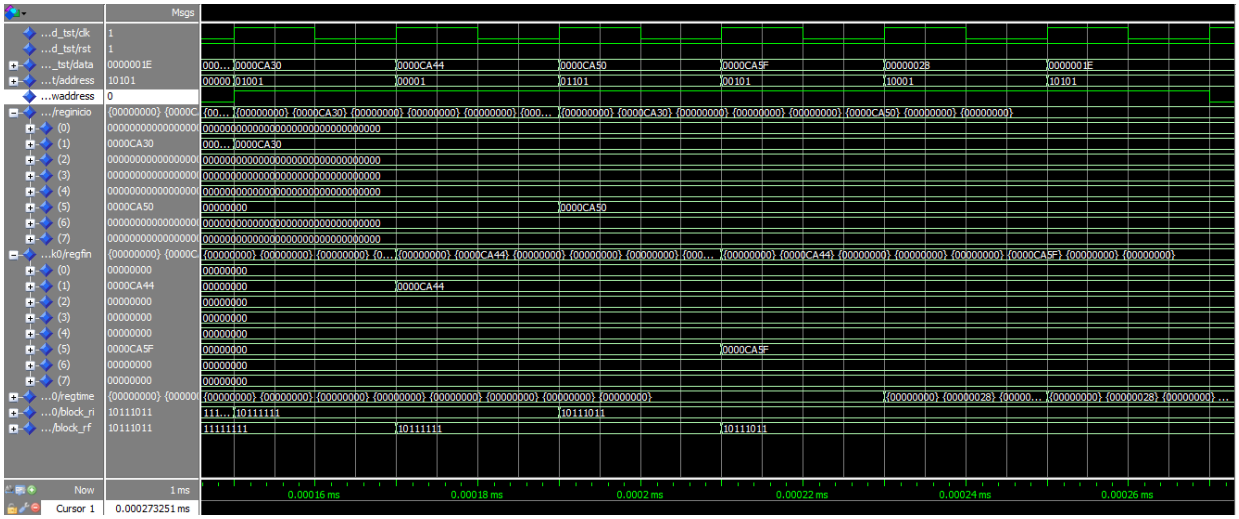

Figura 28: Simulación del módulo aislado (reset y configuración)

En la figura 29 se puede comprobar los resultados de la primera de las versiones del banco de pruebas diseñado. Se ejecutan dos veces cada una de las dos rutinas, de forma que ambas ejecuciones son idénticas y no presentan error de ejecución. Cuando la dirección del bus (avalon_addr) coincide con alguna de los valores almacenados en reginicio y además la señal write se encuentra activa, se pasa el estado rutina y se almacena en el registro pos el índice correspondiente a la rutina que acaba de comenzar. Además, se activa la protección contra escritura de la rutina en curso.

Una vez inicializada la rutina, comienza la cuenta del watchdog y se habilita el espia, de modo que cada vez que se activa la señal newdata se actualiza la firma en el registro result. Cuando en el bus de direcciones aparece la dirección de fin de la rutina en curso y además la señal write está activada, se procede a finalizar la rutina. En primer lugar se pasa al estado fin_rutina en el que se actualiza por última vez la firma con el dato presente en el bus.

A continuación, como se trata de la primera ejecución de la rutina, se pasa al estado iteración_1. En este estado se capta la firma y se almacena en la posición correspondiente del banco de registros, se borra el registro del resultado, y se invierte el bit asignado del registro check.

Finalmente se vuelve al estado de espera donde se inicializa el watchdog.

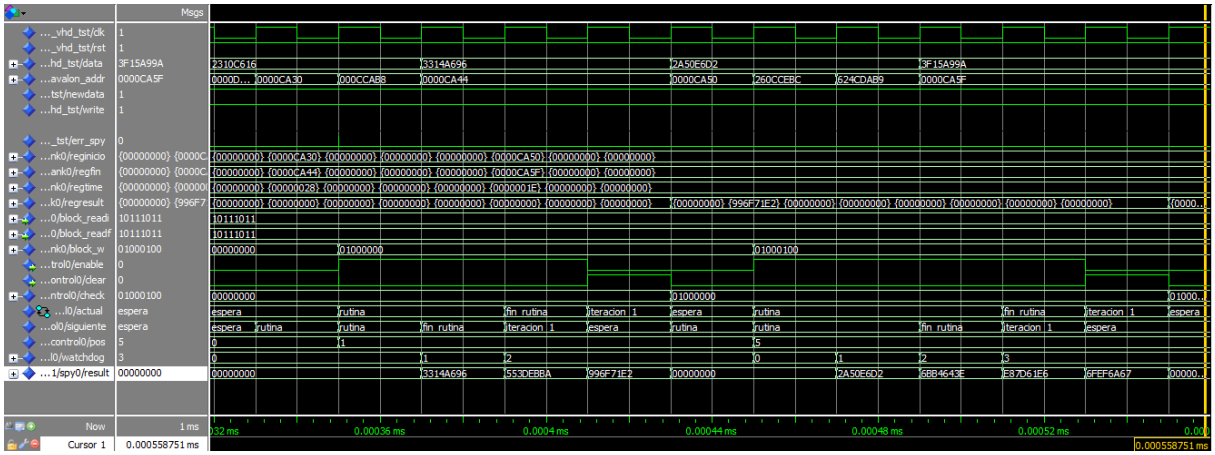

Figura 29: Simulación del módulo aislado (1ª ejecución de las rutinas)

A continuación se ejecutan de nuevo ambas rutinas (figura 30). En este caso, tras el estado fin_rutina se pasa al estado iteración_2, en el cual se compara el resultado de esta nueva ejecución con el almacenado en el banco de registros. Para esta versión del banco de pruebas ambos valores coinciden y por tanto no se activa la señal que indica un error de ejecución (err_spy). Además, se invierte el bit correspondiente del registro check, se desactiva la protección contra escritura de la rutina ejecutada y se borra el registro que contiene la firma para una ejecución posterior.

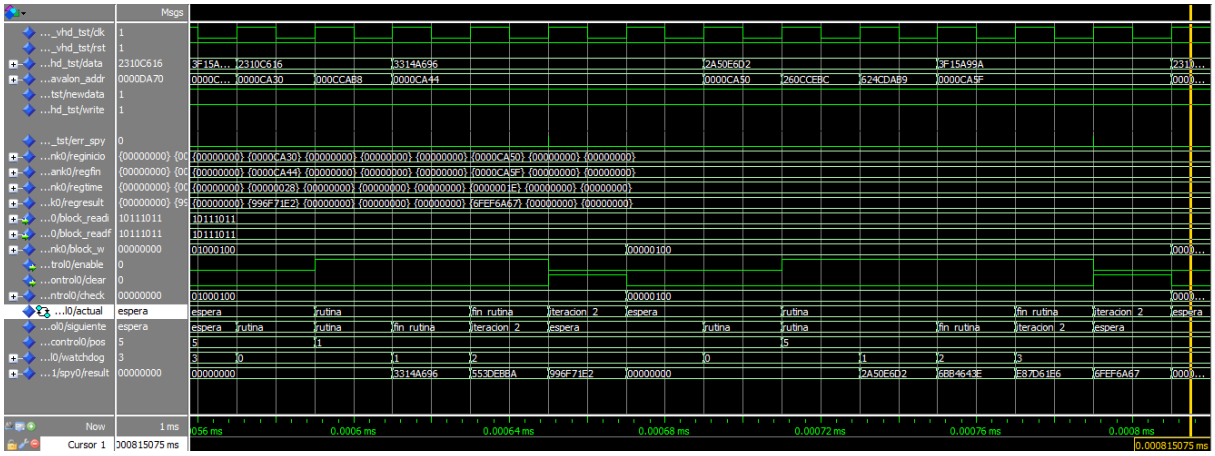

Figura 30: Simulación del módulo aislado (repetición correcta de ambas rutinas)

En la segunda de las versiones diseñadas del banco de pruebas se comprueba que el módulo de detección es capaz de realizar su función y encontrar los errores de ejecución que se producen. En este caso, durante la segunda ejecución de cada una de las rutinas se producirán dos errores diferentes.

En la primera de ellas, el resultado de ambas ejecuciones será diferente. Como se puede observar en la figura 31, el valor presente en el bus de datos (data) difiere respecto al encontrado durante la primera ejecución (figura 29), por lo que la firma generada (result) es distinta al finalizar la rutina. Por tanto, se activa la señal de error de ejecución (err_spy) al término de dicha rutina.

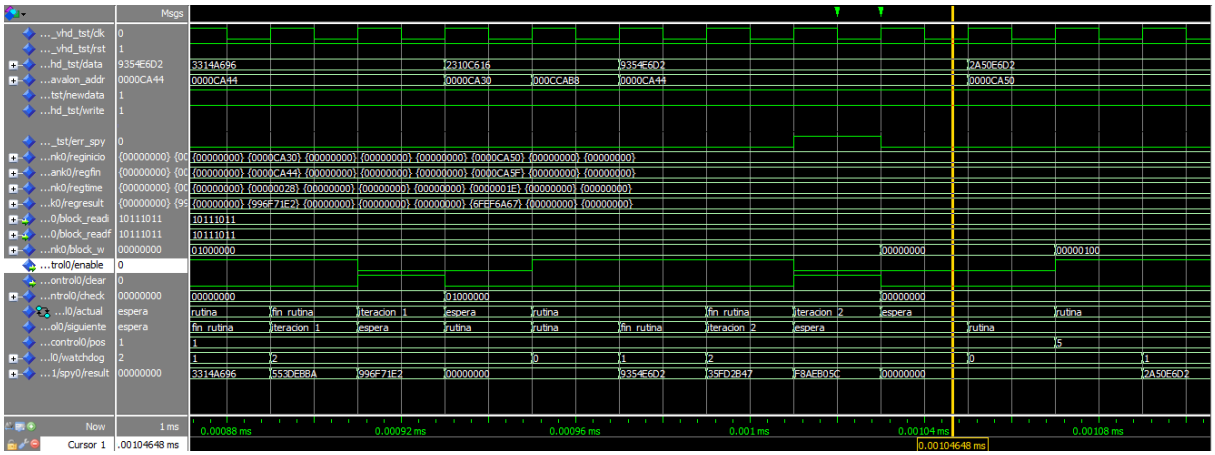

Figura 31: Simulación del módulo aislado (fallo por distintos resultados)

El fallo en la segunda de las rutinas se debe a sobrepasar el número máximo de ciclos de reloj permitidos para su ejecución. La figura 32 muestra cómo una vez el watchdog alcanza el valor límite de finalización de ejecución almacenado en el canco de registros (regtime) correspondiente, se activa la señal de error de ejecución (err_spy), se borra el registro que genera la firma (result) y se retorna al estado de espera. Además se desactiva la protección contra escritura y se pone a 0 el bit del registro check correspondiente a la rutina ejecutada.

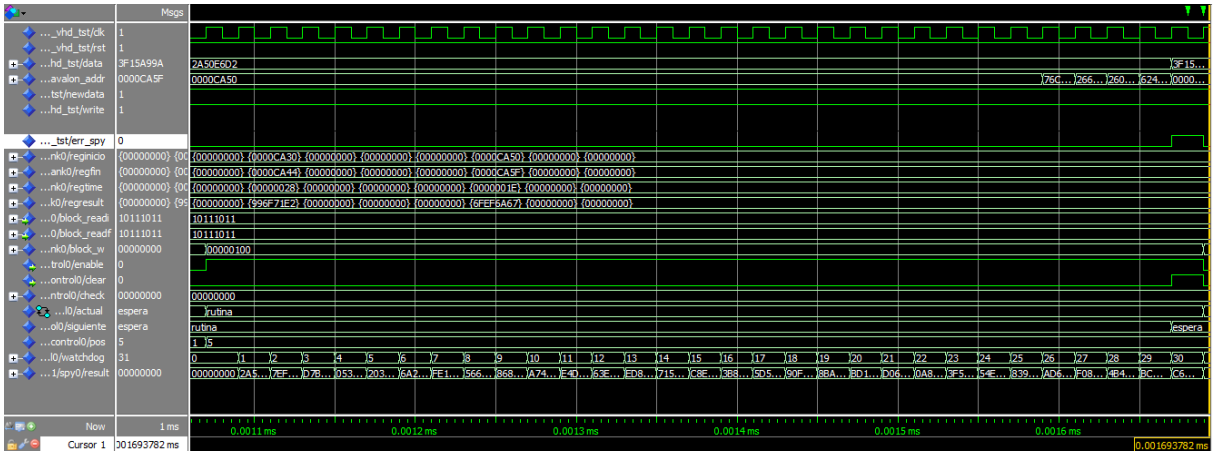

Figura 32: Simulación del módulo aislado (desbordamiento del watchdog)

Por último, la tercera de las versiones del banco de pruebas permite la verificación del funcionamiento de las protecciones contra escritura. En la figura 33 se observan varios intentos de escritura en el banco de registros una vez que el módulo de detección ha iniciado su labor. El primero de los intentos, en la dirección de inicio de la posición número 2, tiene éxito ya que se trata de una región del banco de registros que no está protegida contra escritura, (el bit correspondiente del registro block_w está desactivado).

El segundo intento, en la dirección de fin con el índice 0, no tiene efecto ya que en ese mismo momento se está modificando la protección contra escritura. Se trata de una situación imposible en la práctica; el bus sólo permite comunicar a un maestro con un esclavo por turno, por lo que si el microprocesador se comunica con el módulo de detección no podría comunicarse con el esclavo objetivo de manera simultánea.

Los otros dos intentos de escritura se producen para las posiciones 1 y 5 del banco de registros, zonas protegidas contra escritura. Como puede verse la protección contra escritura impide que se modifique la configuración de una rutina desde que se inicia su primera ejecución hasta que finaliza la segunda o se detecta un error de ejecución.

Figura 33: Simulación del módulo aislado (protección contra escritura)

5.1.2 Simulación de la interfaz

Este paso de depuración del módulo de detección consiste en la simulación del bloque Interfaz conectado al bus Avalon. Para ello se ha realizado una pequeña modificación; se ha eliminado la parte correspondiente al envío de interrupciones cuando el módulo encuentra algún fallo de ejecución, (señales err_spy, ins_spy). En cuanto a la configuración de la longitud de la señal de direcciones (avs_s0_address), se ha considerado un total de 17 bits.

En las figuras 34 y 35 se muestran varias operaciones de escritura tanto en la propia interfaz como en el dispositivo esclavo objetivo.

En el caso de una escritura en el dispositivo de memoria (figura 34), se puede comprobar que al producirse una transferencia entre el maestro y el esclavo objetivo, se activa de manera inmediata la señal write para indicar al bloque de control que existe una transferencia disponible para ser evaluada. Es entonces en el siguiente ciclo de reloj cuando se activa la señal newdata para indicar al bloque espía que hay un nuevo dato en el bus para ser capturado.

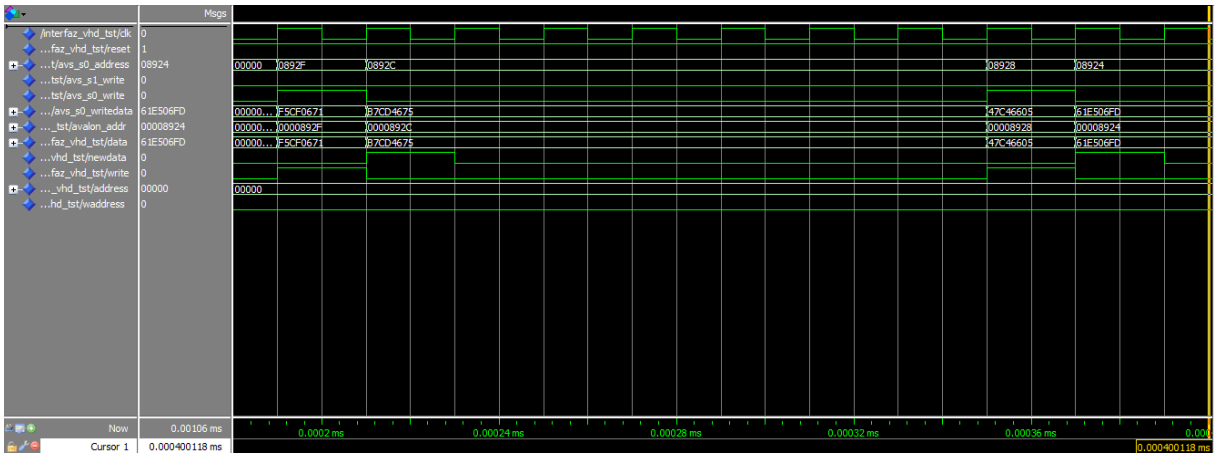

Figura 34: Simulación de la interfaz I (Comunicación con la memoria)

Como se puede ver, tras cada operación de escritura en la interfaz (figura 35) se habilita la escritura de la dirección presente en ese momento en el bus (avs_s0_address) en el correspondiente banco de registros mediante la activación de la señal waddres. Dicha dirección se transfiere al módulo de detección mediante la señal address.

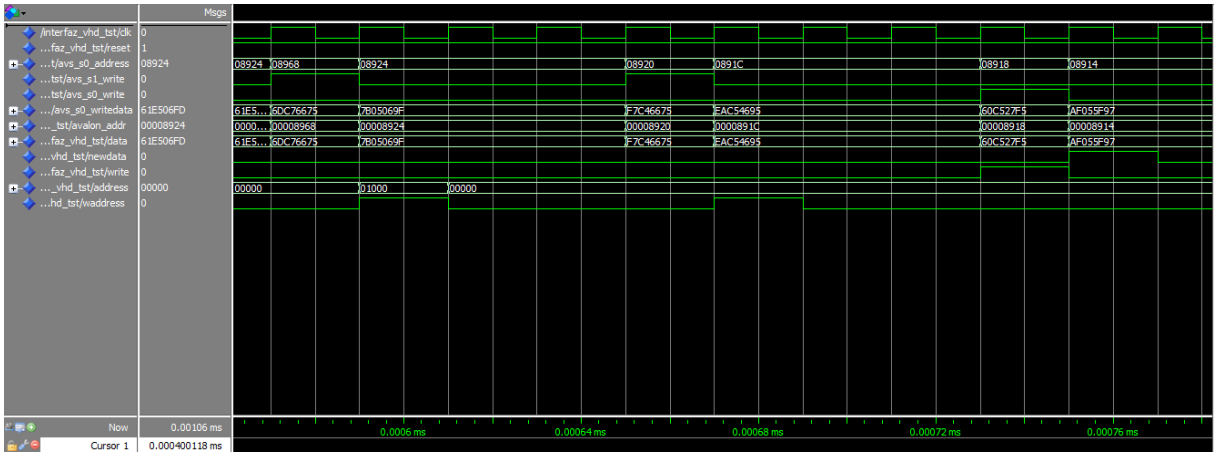

Figura 35: Simulación de la interfaz II (comunicación con la interfaz)

También es importante fijarse en que la interfaz transfiere de manera directa los valores de dirección (avs_s0_address) y datos (avs_s0_writedata) tomados del bus Avalon al resto de bloques mediante las señales avalon_addr y data respectivamente.

5.1.3 Simulación del módulo de detección al completo

En último lugar se ha realizado una simulación del módulo de detección completo conectado al bus Avalon. Lo más interesante de esta prueba es poder comprobar que la comunicación entre el bloque Interfaz y el resto de bloques del módulo de detección se realiza adecuadamente, y que las interrupciones por error de ejecución y de escritura se activan cuando corresponde.

Para esta fase, la configuración del módulo de detección ha sido la siguiente:

- Capacidad para vigilar ocho rutinas independientes, es decir, se ha asignado un valor 3 de addrsize.
- El maestro objetivo y el esclavo objetivo son, respectivamente, el microprocesador Nios II y el dispositivo de memoria.
- El espacio de direcciones asignado es el comprendido entre los valores hexadecimales de 00000 y 7FFFF.

El programa cargado en la memoria del microprocesador Nios II realiza en primer lugar la configuración del módulo de detección para la supervisión de dos rutinas, ubicadas en las posiciones 1 y 7 del banco de registros. Las direcciones de inicio y de fin para la rutina número 1 son, respectivamente, 0D800 y 0D80F. Para la rutina 7, las direcciones correspondientes serán 0D8C0 y 0D8CF, respectivamente. El tiempo máximo de ejecución permitido para la primera rutina es de 10 ciclos y de 15 para la segunda.

En la figura 36 se puede visualizar la configuración de las rutinas que cumplen los requisitos mencionados anteriormente. En dicha imagen puede verse el procedimiento correcto de activación de los registros de dirección de inicio (reginicio) y de su bit de lectura correspondiente (block_ri).

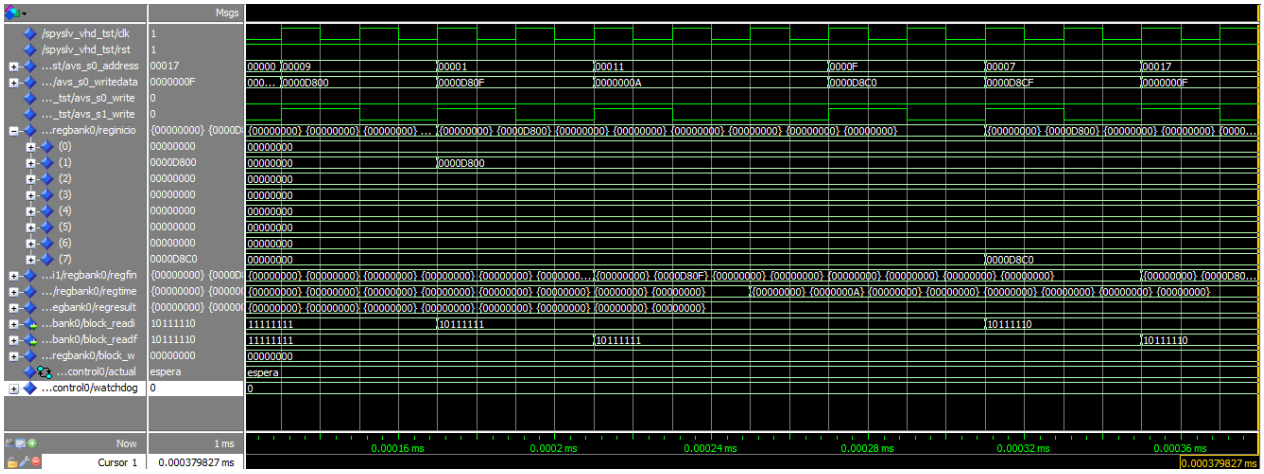

Figura 36: Simulación del módulo de detección completo (configuración de rutinas)

Tras la configuración de las rutinas se realiza una primera ejecución de cada una de ellas para posteriormente realizar en un primer lugar una rutina correcta (figura 37), un intento de escritura en registro protegido (figura 38), un error de ejecución por sobrepasar el tiempo límite máximo para realizar la rutina (figura 39) y un error de ejecución por valor incorrecto en el bus de datos a la finalización de la rutina (figura 40).

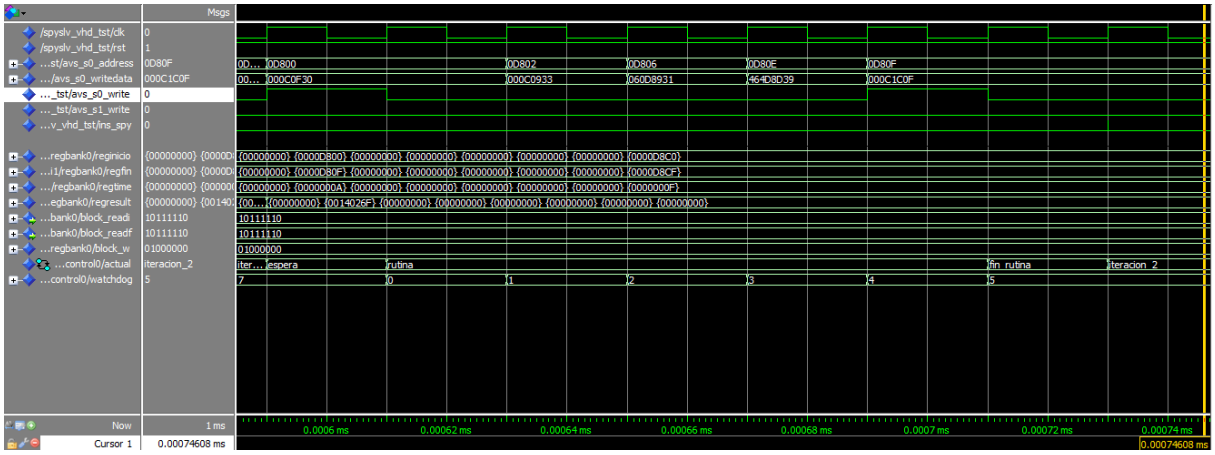

Figura 37: Simulación del módulo de detección completo (ejecución rutina correcta)

En la siguiente figura se puede ver el resultado de un intento de escritura en un registro protegido contra ello. Como se ve en la imagen, no se realiza la escritura del valor en el banco de registros correspondiente ya que se encuentra protegido.

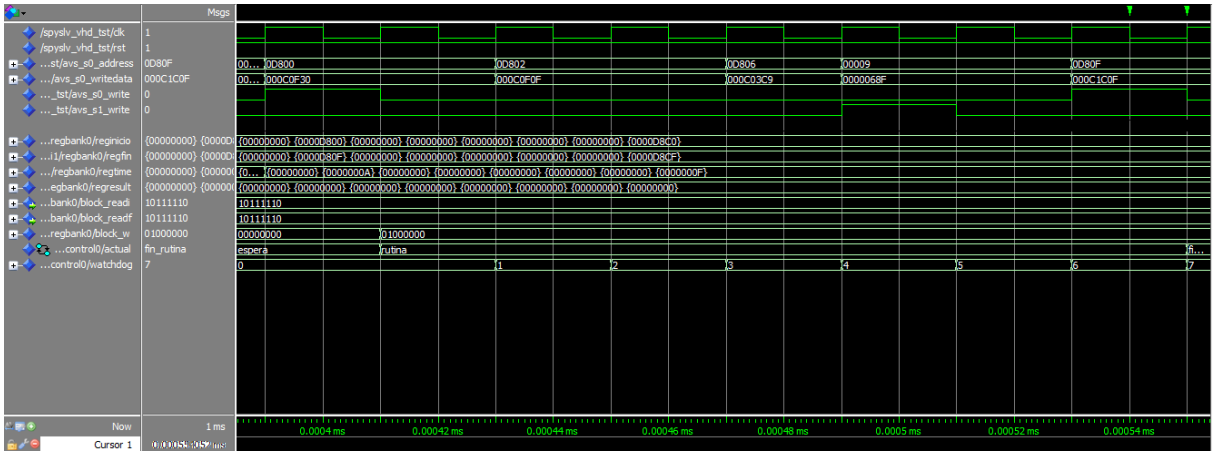

Figura 38: Simulación del módulo de detección completo (intento de escritura en registro protegido)

Por último, en las figuras 39 y 40 se puede observar la activación de una interrupción por error de ejecución del programa (ins_spy). La primera de ellas se debe a sobrepasar el tiempo límite permitido para dicha rutina, mientras que en la segunda es resultado de obtener un valor incorrecto en la finalización de la rutina correspondiente. El tratamiento de la interrupción es el correspondiente al indicado en las rutinas incluidas en el programa de control del procesador.

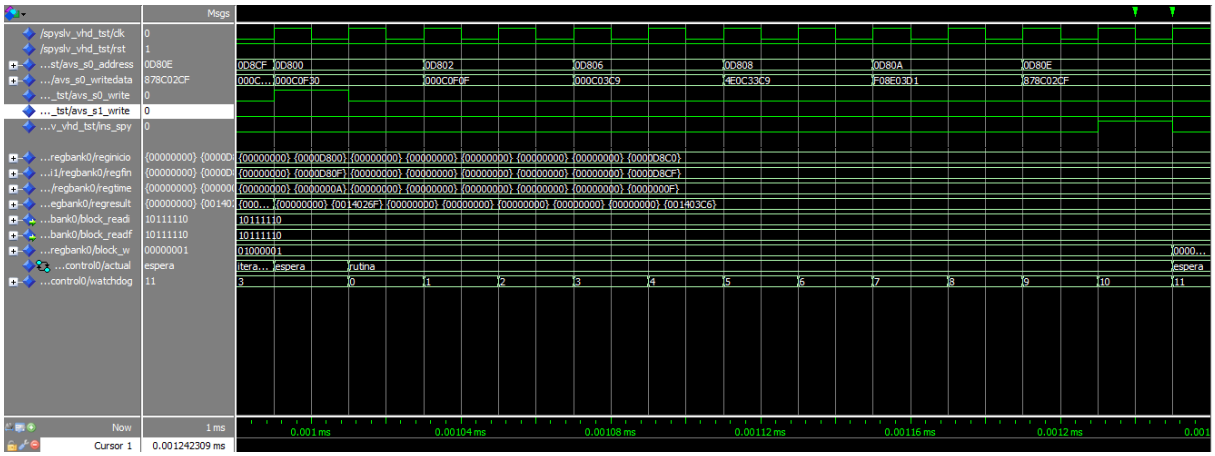

Figura 39: Simulación del módulo de detección completo (interrupción por error de ejecución por desbordamiento del watchdog)

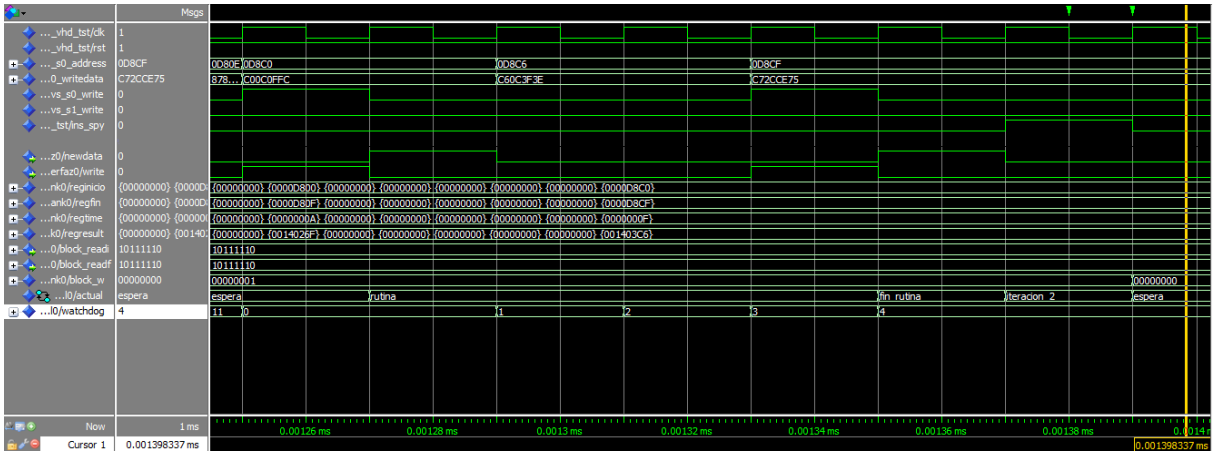

Figura 40: Simulación del módulo de detección completo (interrupción por error de ejecución debido a valor incorrecto al finalizar la rutina)

5.2 Resultado de síntesis

Al realizar la síntesis del diseño podemos conocer el área ocupada por el mismo y su frecuencia máxima de funcionamiento al implementarlo dentro de un dispositivo lógico programable. En este caso se han realizado tres síntesis distintas:

- Una con únicamente el sistema Nios II sin el módulo de detección.
- Dos síntesis del sistema completo con el módulo de detección incorporado configurado para vigilar cuatro y ocho rutinas diferentes respectivamente.

Para la realización de la síntesis del diseño se utilizó el programa Quartus II del fabricante Altera, y se seleccionó una placa de la familia Cyclone III como FPGA de implementación.

Los datos de la configuración seleccionada para el sistema basado en el microprocesador Nios II, son los expuestos en el apartado 3 de la presente memoria y por ello no se volverán a explicar nuevamente aquí.

Los resultados de síntesis expuestos a continuación se componen tanto de ocupación como de análisis de tiempos. Como datos relevantes citaremos:

- Número de ALUTs combinacionales. Una ALUT es un elemento propio de las FPGAs de la familia Altera que consite en la agrupación de dos LUTs, tablas de búsqueda o Look-Up Tables, con el objetivo de aprovechar señales comunes.
- Número de registros.
- Número de bloques DSP, módulos aritméticos.
- Número de bloques M9K, memorias de 9 KB.

La información recopilada tras las diversas síntesis realizadas es la que aparece a continuación:

Diseño	ALUTs	Registros	Bloques DSP	Bloques M9K
Nios II sin módulo de detección	2258	1248	0	2
Nios II con módulo de detección para 4 rutinas	2335 (+3.4%)	1284 (+2.9%)	0	2

A continuación se muestran los resultados obtenidos tras el análisis de tiempos. En este caso se facilita la frecuencia máxima del camino más lento posible para cada uno de los casos.

Diseño	Frecuencia máx. (MHz)
Nios II sin módulo de detección	179.99
Nios II con módulo de detección para 4 rutinas	150.29

6. Conclusiones y trabajos futuros

Como ya se adelantó en el apartado de objetivos a conseguir, la finalidad del presente proyecto era la adaptación de un módulo IP de detección de fallos para un sistema basado en un microprocesador Nios II. La principal tarea a desarrollar era la determinación de la existencia de errores transitorios mediante la supervisión de las comunicaciones del bus interno.

En este caso únicamente se seleccionó una configuración para la supervisión de las comunicaciones entre el microprocesador y un dispositivo de memoria, sin embargo, tal y como se menciona de manera reiterada en los diversos apartados que componen la memoria, esta aplicación puede extenderse a la supervisión de comunicaciones entre un maestro y un esclavo cualquiera, tanto en transferencias de escritura como en transferencias de lectura.

Entre uno de los aspectos más positivos a valorar de los resultados obtenidos es la versatilidad de funcionalidad que ofrecen todas las herramientas asociadas al desarrollo de sistemas basados en el microprocesador Nios II. El fabricante Altera proporciona todo un paquete completo de herramientas dedicadas y optimizadas para facilitar la labor del desarrollador en todos los niveles del proyecto.

La aplicación Quartus II permite un diseño de lógica basado en sistema CAD o HDL para la elaboración de módulos externos al sistema Nios II que proporcionan la oportunidad de realizar múltiples configuraciones de sistemas que funcionen bajo un mismo dispositivo lógico programable. Esto supone una flexibilidad prácticamente total en el ciclo de trabajo a la hora de diseñar, verificar y rediseñar en función de los resultados obtenidos, sin necesidad de cambiar de soporte o configuración global.

La herramienta Qsys concede una manera simple, eficaz y completa para la configuración del sistema Nios II. Una interfaz gráfica intuitiva proporciona al desarrollador llevar a cabo complejos diseños sin necesidad de conocer en profundidad la arquitectura interna del microprocesador. La posibilidad de añadir módulos propios al sistema, o la presencia de una extensa biblioteca de componentes para diversos objetivos, amplia aún más las ventajas del uso de este tipo de sistema.

Para finalizar es importante mencionar que debido a ser un sistema desarrollado por el propio fabricante, está completamente optimizado para funcionar en sus propias FPGAs, lo que supone, como se ha podido observar en los resultados de síntesis, que su rendimiento en aspectos de área ocupada son sensiblemente mejores que otros sistemas análogos presentes en el mercado actual.

Como posibles líneas de trabajo a seguir en un futuro se pueden citar:

- Desde el lado del diseño del módulo IP: podría considerarse que partiendo de la versión actual desarrollada y conociendo sus resultados en sistemas implementados, se realizase un análisis de rendimiento con el objeto de buscar optimizaciones o ampliaciones de funcionalidad que permitan mejorar el coste económico, el área ocupada en el dispositivo o su velocidad de funcionamiento para una misma aplicación.

- Desde el lado de implementación del módulo IP en sistema basados en interfaces tipo Avalon: Sería interesante plantearse un nuevo diseño configurable que permita agrupar todas las señales presentes en dicho estándar, permitiendo de esa manera una implementación global a cualquier sistema desarrollado sin la necesidad de tener que realizar una configuración manual del módulo en cuanto a señales y tratamiento se refiere.

- Desde un punto de vista general: la realización de un estudio de sistemas análogos diseñados para diversas arquitecturas que utilicen microprocesadores o buses de comunicaciones diferentes, con la finalidad de contrastar resultados.

Bibliografía

[1] Nios II Processor Reference Handbook, Altera Corporation, 2010.

[2] Embedded Design Handbook, Altera Corporation, 2011.

[3] SOPC Builder User Guide, Altera Corporation, 2010.

[4] Nios II Software Developer's Handbook, 2011.

[5] Avalon Interface Specification, Altera Corporation, 2011.

[6] Quartus II Handbook v12.0, Altera Corporation, 2012.

[7] ARM, AMBA Specification (Rev 2.0), ARM IHI 0011A, 1999.

[8] AHB to Avalon & Avalon to AHB Bridges White Paper, Altera Corporation, 2003.

[9] A. J. Sánchez Clemente, "Diseño de un módulo I-IP para la detección de errores transitorios en sistemas embebidos", Enero 2011.

Anexo A. Presupuesto

A.1 Introducción

El objetivo del presente proyecto es el diseño y adaptación de un módulo capaz de detectar errores en sistemas embebidos basados en el microprocesador Nios II mediante la supervisión del bus de comunicaciones Avalon del sistema. El módulo diseñado debe ser económico tanto en coste como en espacio ocupado, debe afectar en la menor medida posible a las prestaciones del microprocesador al cual se conecta y debe poderse insertar en el sistema de forma no intrusiva.

A.2 Fases del proyecto

La ejecución del proyecto se divide en las fases que se detallan a continuación:

- Preparación previa. Se refiere al estudio del protocolo de comunicaciones Avalon, de la arquitectura del microprocesador Nios II y de las herramientas que llevan asociadas. También se incluye una ligera revisión del protocolo AMBA y de sus posibles adaptaciones respecto al bus Avalon. Se estiman unas 280 horas de dedicación.

- Diseño, adaptación y depuración del módulo de detección. Se estima un total de 250 horas dedicadas.

- Pruebas. Por un lado se realiza la síntesis del diseño con el objetivo de conocer los requerimientos de espacio y frecuencia de funcionamiento. Por otro, se efectúa una serie de simulaciones de verificación para comprobar que los procesos previstos ante los diferentes casos de detección de errores en rutinas son los esperados. El tiempo total invertido para esta fase se estima en 100 horas dedicadas.

A.3 Desglose de costes

Los costes directos asociados al proyecto constan de tres conceptos: personal, equipos y licencias software.

Para el cálculo de los costes de personal se considera un coste horario del empleado de 25 € por hora, una jornada laboral de 8 horas y 5 días laborables a la semana. Se considera también que el proyecto ha sido realizado únicamente por un solo empleado.

Se necesita un puesto informático con un equipo disponible durante toda la duración del proyecto. La valoración de dicho equipo se estipula en 2000 €, y se le atribuye un período de amortización de 3 años.

Durante el desarrollo del proyecto es necesario utilizar los programas informáticos Modelsim (para la depuración y validación del sistema) y Quartus II (para la síntesis). El coste anual de las licencias de estos programas se estiman en 8000 € y 6000 € al año respectivamente. Estos programas se emplean en diversos proyectos de la misma empresa, por lo que se imputa un coste proporcional al número de horas dedicadas al presente proyecto.

Se estima que los costes indirectos del proyecto suponen un 10% de los costes directos, además, la empresa aplica una tasa del 35% del coste total como beneficios.

En la siguiente tabla se refleja el coste imputado a cada uno de los conceptos antes mencionados, así como el coste total del proyecto:

Concepto	**Cantidad (€)**
Costes de personal	13500
Amortización de equipos	210
Licencias software	1045
Costes indirectos	1749
Total	16504

El presupuesto completo del presente proyecto se cifra en 22280 €.

Anexo B. Código fuente de los archivos del proyecto

B.1 mytypes.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package mytypes is
type banco_reg is array (integer range <>) of std_logic_vector(31 downto 0);
end package mytypes;
```

B.2 Spyslv.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.mytypes.all;

entity spyslv is
  generic (
        addrsize: integer := 3;
                                -- numero de bits para direccionar el banco de registros-
                                resultados, el numero de rutinas soportadas sera 2**adrrsize
        addrwidth       :  integer := 17);
  port (
-- señales generales
        rst                      : in  std_logic;
        clk                      : in  std_logic;
-- señales esclavo Avalon
        avs_s0_write             : in  std_logic;
-- señal de escritura general
        avs_s0_address           : in      std_logic_vector (addrwidth-1 downto 0);
-- señal de direccion
        avs_s0_writedata         : in      std_logic_vector (31 downto 0);
-- señal de datos de escritura
        avs_s1_write             : in  std_logic;
-- señal de escritura en el modulo
        avs_s1_writedata         : in      std_logic_vector (31 downto 0);
                                                -- señal vacia (no se usa)
```

```
-- señales de interrupcion
        ins_spy                 : out std_logic
                                        -- señal de interrupcion para error de ejecucion
   );
end;

architecture spyslv of spyslv is

-- declaracion de componentes

component interfaz is
generic(
        addrsize        : integer := 3;
        addrwidth       :  integer := 17);
                                        -- Anchura de direcciones en el esclavo

port (
-- señales generales
        reset           : in  std_logic;
        clk             : in  std_logic;
-- señales esclavo Avalon
        avs_s0_write  : in  std_logic;
                                                -- señal de escritura general
        avs_s0_address : in    std_logic_vector (addrwidth-1 downto 0);
                                                -- señal de direccion
        avs_s0_writedata : in std_logic_vector (31 downto 0);
                                                -- señal de datos de escritura
        avs_s1_write  : in  std_logic;
                                                -- señal de escritura en el modulo
        avs_s1_writedata : in std_logic_vector (31 downto 0);
                                                 -- señal vacia (no se usa)
-- señales de interrupcion
        ins_spy         : out std_logic;
                                                -- señal de interrupcion para error de ejecucion
-- señales modulo deteccion errores
        err_spy         : in     std_logic;
                                                -- error de ejecucion
        newdata         : out std_logic;
                                                -- nuevo dato en el bus
        data            : out std_logic_vector(31 downto 0);
                                                -- bus de datos
        avalon_addr     : out std_logic_vector(31 downto 0);
```

```
                                        -- direccion del bus avalon para el control
        write           : out std_logic;
                                        -- escritura del maestro al esclavo
        address         : out std_logic_vector(addrsize+1 downto 0);
                                        -- direccion del banco de registros
        waddress        : out std_logic
                                        -- habilitar escritura de direcciones
                );
end component;
component regbank is
generic (
        addrsize : integer := 3);
                                        -- numero de bits para direccionar el banco de
                                        registros-resultados, el numero de rutinas
                                        soportadas sera 2**adrrsize
port (
        data    : in std_logic_vector(31 downto 0);
                                        -- bus de datos, proveniente de la interfaz
        address : in std_logic_vector(addrsize+1 downto 0);
                                        -- direccion del banco de registros,
                                        proveniente de la interfaz
        waddress : in std_logic;
                                        -- habilitar escritura de direcciones
        rst     : in std_logic;
        clk     : in std_logic;
        resultado : in std_logic_vector(31 downto 0);
                                        -- resultado
        raddress : in integer range 0 to (2**addrsize)-1;
                                        -- direccion del banco de registros de
                                        resultado, proveniente del control
        wresult : in std_logic;
                                        -- habilitar escritura de resultado
        block_write     : in std_logic;
                                        -- proteccion contra escritura mientras se
                                        comprueba una rutina, indicado por control
        block_en        : in std_logic;
                                        -- habilitar modificacion de proteccion contra
                                        escritura, indicado por control
        iniaddress      : out banco_reg (0 to (2**addrsize)-1);
                                        -- direccion de inicio de todos los intervalos
        outaddress      : out std_logic_vector(31 downto 0);
                                        -- direccion de fin del intervalo actual
```

```
        timer   : out std_logic_vector(31 downto 0);
                                        -- salida del watchdog-timer hacia el control
        firstresult : out std_logic_vector(31 downto 0);
                                        -- resultado de la primera iteracion del
                                        intervalo actual
        block_readi     : out std_logic_vector (0 to (2**addrsize)-1);
                                         -- proteccion contra lectura, direccion inicio
                                        => '1'
        block_readf     : out std_logic_vector (0 to (2**addrsize)-1)
                                        -- proteccion contra lectura, direccion fin
                                         => '1'
                );
end component;
component control is
generic (
        addrsize        : integer := 3);
                                        -- numero maximo de rutinas soportadas
port (
        result_spy      : in std_logic_vector(31 downto 0);
                                         -- resultado proveniente del espia
        rst             : in std_logic;
        clk             : in std_logic;
        avalon_addr     : in std_logic_vector(31 downto 0);
                                         -- direccion del bus amba ahb proveniente de
                                        la interfaz
         write          : in std_logic;
                                        -- escritura del maestro al esclavo
        iniaddress      : in banco_reg (0 to (2**addrsize)-1);
                                         -- direccion de inicio de todos los intervalos
        outaddress      : in  std_logic_vector(31 downto 0);
                                        -- direccion de fin del intervalo actual
        timer           : in std_logic_vector(31 downto 0);
                                        -- salida del watchdog-timer hacia el control
        firstresult     : in std_logic_vector(31 downto 0);
                                         -- resultado de la primera iteracion del
                                        intervalo actual
        block_readi     : in std_logic_vector (0 to (2**addrsize)-1);
                                        -- proteccion contra lectura, direccin inicio =>
                                        '1'
        block_readf     : in std_logic_vector (0 to (2**addrsize)-1);
                                        -- proteccion contra lectura, direccin fin
                                         => '1'
```

```
        err_spy         : out std_logic;
                                        -- error de ejecucion
        enable          : out std_logic;
                                        -- habilitacion, en el rango de direcciones de
                                        algun bloque que se debe espiar
        clear           : out std_logic;
                                        -- borra el resultado, indicado por control
        result_regbank : out std_logic_vector(31 downto 0);
                                        -- resultado hacia el banco de registros
        raddress        : out integer range 0 to (2**addrsize)-1;
                                        -- direccion del banco de registros de resultado
        wresult         : out std_logic;
                                         -- habilitar escritura de resultado
        block_write     : out std_logic;
                                        -- proteccion contra escritura mientras se
                                        comprueba una rutina, indicado por control
        block_en                        : out std_logic
                                        -- habilitar modificacion de proteccion contra
                                        escritura, indicado por control
    );
end component;

component spy is

port (
        enable          : in std_logic;
                                        -- habilitacion, en el rango de direcciones de
                                        algun bloque que se debe espiar
        clear           : in std_logic;
                                        -- borra el resultado, indicado por control
        rst             : in  std_logic;
        clk             : in  std_logic;
        newdata         : in std_logic;
                                        -- nuevo dato en el bus
        data            : in std_logic_vector(31 downto 0);
                                        -- bus de datos, proveniente de la interfaz
        resultado       : out std_logic_vector(31 downto 0)
                                        -- se pasa el resultado al control
                );
end component;

-- declaracion de señales
```

```
signal avalon_addr    : std_logic_vector(31 downto 0);
                                -- direccion del bus amba avalon para el
                                control
signal newdata        : std_logic;
                                -- nuevo dato en el bus
signal data           : std_logic_vector(31 downto 0);
                                -- bus de datos
signal address        : std_logic_vector(addrsize+1 downto 0);
                                -- direccion del banco de registros
signal waddress       : std_logic;
signal wresult        : std_logic;
                                -- habilitar escritura de resultado
signal raddress       : integer range 0 to (2**addrsize)-1;
                                -- direccion del banco de registros de
                                resultado, proveniente del control
signal iniaddress     : banco_reg (0 to (2**addrsize)-1);
                                -- direccion de inicio de todos los intervalos
signal outaddress     : std_logic_vector(31 downto 0);
                                -- direccion de fin del intervalo actual
signal timer          : std_logic_vector(31 downto 0);
                                -- salida del watchdog-timer hacia el control
signal firstresult    : std_logic_vector(31 downto 0);
                                --resultado de la primera iteracion del intervalo
                                actual
signal result_spy     : std_logic_vector(31 downto 0);
                                -- resultado proveniente del espia
signal result_regbank : std_logic_vector(31 downto 0);
                                -- resultado hacia el banco de registros
signal enable         : std_logic;
                                -- habilitacion, en el rango de direcciones de
                                algun bloque que se debe espiar
signal clear          : std_logic;
                                -- borra el resultado, indicado por control
signal block_write    : std_logic;
                                -- proteccion contra escritura mientras se
                                comprueba una rutina, indicado por control
signal block_en       : std_logic;
                                -- habilitar modificacion de proteccion contra
                                escritura, indicado por control
signal err_spy        : std_logic;
                                -- error de ejecucion
signal block_readi    : std_logic_vector ((2**addrsize)-1 downto 0);
```

```
        signal block_readf      : std_logic_vector ((2**addrsize)-1 downto 0);
        signal write            : std_logic;

begin

interfaz0:      interfaz generic map(addrsize,addrwidth)
        port map(
rst,clk,avs_s0_write,avs_s0_address,avs_s0_writedata,avs_s1_write,avs_s1_writedata,ins_
spy,err_spy,newdata,data,avalon_addr,write,address,waddress);

regbank0:       regbank generic map(addrsize)
        port map(
data,address,waddress,rst,clk,result_regbank,raddress,wresult,block_write,block_en,iniaddr
ess,outaddress,timer,firstresult,block_readi,block_readf);

control0:       control generic map (addrsize)
        port map(
result_spy,rst,clk,avalon_addr,write,iniaddress,outaddress,timer,firstresult,block_readi,bloc
k_readf,err_spy,enable,clear,result_regbank,raddress,wresult,block_write,block_en);

spy0:           spy
        port map( enable,clear,rst,clk,newdata,data,result_spy);

end spyslv;
```

B.3 Interfaz.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity interfaz is
generic(
        addrsize        :       integer := 3;
        addrwidth       :  integer := 17);
                                        -- Anchura de direcciones en el esclavo

port (
-- señales generales
        reset           : in  std_logic;
        clk             : in  std_logic;
-- señales esclavo Avalon
```

```
        avs_s0_write  : in  std_logic;
                                                -- señal de escritura general
        avs_s0_address : in    std_logic_vector (addrwidth-1 downto 0);
                                                -- señal de direccion
        avs_s0_writedata : in std_logic_vector (31 downto 0);
                                                -- señal de datos de escritura
        avs_s1_write  : in  std_logic;
                                                -- señal de escritura en el modulo
        avs_s1_writedata : in std_logic_vector (31 downto 0);
                                                -- señal vacia (no se usa)
-- señal de interrupcion
        ins_spy         : out std_logic;
                                                -- señal de interrupcion para error de ejecucion
-- señales modulo deteccion errores
        err_spy         : in     std_logic;
                                                -- error de ejecucion
        newdata         : out std_logic;
                                                -- nuevo dato en el bus
        data            : out std_logic_vector(31 downto 0);
                                                -- bus de datos
        avalon_addr    : out std_logic_vector(31 downto 0);
                                                 -- direccion del bus avalon para el control
        write           : out std_logic;
                                                -- escritura del maestro al esclavo
        address         : out std_logic_vector(addrsize+1 downto 0);
                                                -- direccion del banco de registros
        waddress        : out std_logic
                                                -- habilitar escritura de direcciones
                );
end;

architecture interfaz of interfaz is

-- señales internas
        signal newtrans : std_logic;
                                                 -- nueva transferencia
        signal s_address : std_logic_vector(addrsize+1 downto 0);
                                                 -- direccion del banco de registros
        signal s_waddress : std_logic;
                                                -- habilitacion de escritura en registro
        signal newtrans_c : std_logic;
```

```
                                        -- escritura del maestro al esclavo
                                        seleccionados

begin

-- espionaje del bus - detección de nueva transferencia
        process (avs_s0_write)
        begin

                if avs_s0_write = '1' then  -- transferencia de escritura desde el procesador a
un esclavo
                        newtrans_c <= '1';
                else      -- transferencia de lectura desde esclavo al procesador
                        newtrans_c <= '0';
                end if;

        end process;

        process(clk,reset)
        begin
                if reset = '0' then
                        newtrans <= '0';
                elsif clk'event and clk = '1' then
                        newtrans <= newtrans_c;
                        address <= s_address;
                        waddress <= s_waddress;
                end if;
        end process;

--interfaz de comunicaciones
        process (clk,reset)
        begin
                if reset = '0' then  -- se resetean todas las señales relacionadas con el bus
avalon y el estado de transferencia
                        s_address <= (others => '0');
                        s_waddress <= '0';
                elsif clk'event and clk = '1' then
                        s_address <= (others => '0');
                        s_waddress <= '0';
                        if avs_s1_write = '1' then
                                        -- transferencia de escritura entre el procesador
                                        y el modulo de deteccion
```

```
                        s_address <= avs_s0_address(addrsize+1 downto 0);
                        s_waddress <='1';
                end if;
        end if;

    end process;

-- asignacion de interrupciones
    process (err_spy)

    begin
        if err_spy = '1' then
            ins_spy <= '1';
        else
            ins_spy <= '0';
        end if;
    end process;


-- asignacion de salidas
    avalon_addr(addrwidth-1 downto 0) <= avs_s0_address;
    avalon_addr(31 downto addrwidth) <= (others => '0');
    data <= avs_s0_writedata;
    newdata <= newtrans;
    write <= newtrans_c;

end interfaz;
```

B.4 Regbank.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.mytypes.all;
entity regbank is

generic (
    addrsize        : integer := 3);
                        -- numero de bits para direccionar el banco de
                        registros-resultados, el numero de rutinas soportadas
                        sera 2**adrrsize
port (
```

```
        data            : in std_logic_vector(31 downto 0);
                                -- bus de datos, proveniente de la interfaz
        address         : in std_logic_vector(addrsize+1 downto 0);
                                -- direccion del banco de registros, proveniente de la
                                interfaz
        waddress        : in std_logic;
                                -- habilitar escritura de direcciones
        rst             : in std_logic;
        clk             : in std_logic;
        resultado       : in std_logic_vector(31 downto 0);
                                -- resultado
        raddress        : in integer range 0 to (2**addrsize)-1;
                                -- direccion del banco de registros de resultado,
                                proveniente del control
        wresult         : in std_logic;
                                -- habilitar escritura de resultado
        block_write     : in std_logic;
                                -- proteccion contra escritura mientras se comprueba
                                una rutina, indicado por control
        block_en        : in std_logic;
                                -- habilitar modificacion de proteccion contra
                                escritura, indicado por control
        iniaddress      : out banco_reg (0 to (2**addrsize)-1);
                                -- direccion de inicio de todos los intervalos
        outaddress      : out std_logic_vector(31 downto 0);
                                -- direccion de fin del intervalo actual
        timer           : out std_logic_vector(31 downto 0);
                                -- salida del watchdog-timer hacia el control
        firstresult     : out std_logic_vector(31 downto 0);
                                -- resultado de la primera iteracion del intervalo actual
        block_readi     : out std_logic_vector (0 to (2**addrsize)-1);
                                -- proteccion contra lectura, direccion inicio => '1'
        block_readf     : out std_logic_vector (0 to (2**addrsize)-1)
                                -- proteccion contra lectura, direccion fin
                                => '1'
            );
end;

architecture regbank of regbank is

-- banco de registros
```

```
-- por cada rutina soportada hay 4 registros: direcciones de inicio y de fin, valor al que
desborda el watchdog y resultado de la primera ejecución
        signal reginicio :  banco_reg (0 to (2**addrsize)-1);
        signal regfin    :  banco_reg (0 to (2**addrsize)-1);
        signal regtime   :  banco_reg (0 to (2**addrsize)-1);
        signal regresult :  banco_reg (0 to (2**addrsize)-1);
-- bits de protección, uno para cada una de las rutinas soportadas
        signal block_ri          :      std_logic_vector (0 to (2**addrsize)-1);
                                        -- proteccion contra lectura, direccin inicio => '1'
        signal block_rf          :      std_logic_vector (0 to (2**addrsize)-1);
                                        -- proteccion contra lectura, direccin fin                => 
                                        '1'
        signal block_w   :  std_logic_vector (0 to (2**addrsize)-1);
                                        -- proteccion contra escritura => '1'
begin
        -- entrada de datos
        process(clk,rst)
                variable indice : integer range 0 to (2**addrsize)-1;
        begin
                if rst='0' then
                        for i in 0 to (2**addrsize)-1 loop
                                reginicio(i) <= (others => '0');
                                regfin(i)    <= (others => '0');
                                regtime(i)   <= (others => '0');
                                regresult(i) <= (others => '0');
                                block_ri(i)      <= '1';
                                block_rf(i)      <= '1';
                                block_w(i)       <= '0';
                        end loop;
                elsif clk'event and clk='1' then
                        if block_en = '1' then
                        -- modificar proteccion contra escritura
                                block_w(raddress) <= block_write;
                        elsif waddress = '1' then
                                        -- escritura en el bloque de direcciones, si lo permite
                                        la proteccion contra escritura
                                indice := to_integer(unsigned(address(addrsize-1 downto
0)));
                                if block_w(indice) = '0' then
                                        -- proteccion contra escritura desactivada
                                        if address(addrsize+1) = '1' then
                                        -- escritura del tiempo del watchdog-timer
```

```
                                        regtime(indice)<= data;
                                    else
                                        if address(addrsize) = '1' then
                                    -- escritura de la direccion de inicio
                                            reginicio(indice)<= data;
                                            block_ri(indice) <= '0';
                                    -- proteccion contra lectura desactivada
                                        elsif address(addrsize) = '0' then
                                    -- escritura de la direccion de fin
                                            regfin(indice)<= data;
                                            block_rf(indice) <= '0';
                                    -- proteccion contra lectura desactivada
                                        end if;
                                    end if;
                                end if;
                            end if;
                            if wresult = '1' then
                            -- escritura en el bloque de resultados
                                regresult(raddress)<= resultado;
                            end if;
                        end if;
                end process;

                -- salida de datos
                process(reginicio,regfin,regresult,raddress)
                begin
                        iniaddress <= reginicio;
                        outaddress <= regfin(raddress);
                        timer <= regtime(raddress);
                        firstresult <= regresult(raddress);
                        block_readi <= block_ri       ;
                        block_readf <= block_rf       ;
                end process;

end regbank;
```

B.5 Control.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.mytypes.all;


entity control is
 generic (
        addrsize : integer := 3);
                                                -- numero maximo de rutinas soportadas
  port (
        result_spy      : in std_logic_vector(31 downto 0);
                                                 -- resultado proveniente del espia
        rst             : in std_logic;
        clk             : in std_logic;
        avalon_addr     : in std_logic_vector(31 downto 0);
                                                -- direccion del bus amba ahb proveniente de
                                                la interfaz
        write           : in std_logic;
                                                -- escritura del maestro al esclavo
        iniaddress      : in banco_reg (0 to (2**addrsize)-1);
                                                 -- direccion de inicio de todos los intervalos
        outaddress      : in  std_logic_vector(31 downto 0);
                                                 -- direccion de fin del intervalo actual
        timer           : in std_logic_vector(31 downto 0);
                                                 -- salida del watchdog-timer hacia el control
        firstresult     : in std_logic_vector(31 downto 0);
                                                -- resultado de la primera iteracion del
                                                intervalo actual
        block_readi     : in std_logic_vector (0 to (2**addrsize)-1);
                                                -- proteccion contra lectura, direccin inicio =>
                                                '1'
        block_readf     : in std_logic_vector (0 to (2**addrsize)-1);
                                                -- proteccion contra lectura, direccin fin
                                                 => '1'
        err_spy          : out std_logic;
                                                 -- error de ejecucion
        enable          : out std_logic;
```

```
                                        -- habilitacion, en el rango de direcciones de
                                        algun bloque que se debe espiar
        clear           : out std_logic;
                                        -- borra el resultado, indicado por control
        result_regbank : out std_logic_vector(31 downto 0);
                                         -- resultado hacia el banco de registros
        raddress        : out integer range 0 to (2**addrsize)-1;
                                         -- direccion del banco de registros de
                                        resultado
        wresult         : out std_logic;
                                         -- habilitar escritura de resultado
        block_write     : out std_logic;
                                        -- proteccion contra escritura mientras se
                                        comprueba una rutina, indicado por control
        block_en        : out std_logic
                                        -- habilitar modificacion de proteccion contra
                                        escritura, indicado por control
    );

end;

architecture control of control is

        type estado is (espera, rutina, fin_rutina, iteracion_1, iteracion_2);
        signal actual, siguiente : estado;
        signal check            : std_logic_vector (0 to (2**addrsize)-1);
                                        -- 1 iteracion '0' o 2 iteracion '1' para cada
                                        rutina
        signal checki           : std_logic;
                                        -- toma el valor de uno de los bits de check
                                        para operar en la parte combinacional
        signal pos_intervalo : integer range 0 to (2**addrsize)-1;
                                        -- posicion de la rutina actual (combinacional)
        signal pos_en           : std_logic;
                                        -- habilitar registro de posicion
        signal pos              : integer range 0 to (2**addrsize)-1;
                                        -- posicion de la rutina actual (secuencial)
        signal error_spy        : std_logic;
        signal watchdog         : unsigned (31 downto 0);
                                        -- temporizador

begin
```

```
process(actual,iniaddress,outaddress,firstresult,result_spy,avalon_addr,pos,watchdog,write)
    variable indice                 : integer range 0 to (2**addrsize)-1;
    variable coincidencia : std_logic;
  begin
      enable <= '0';
      clear <= '0';
      error_spy <= '0';
      wresult <= '0';
      result_regbank <= (others => '0');
      block_write <= '0';
      block_en <= '0';
      pos_en <= '0';
      pos_intervalo <= pos;
      checki <= check(pos);
    case (actual) is
      when espera =>
            coincidencia := '0';
            indice := 0;
            for i in 0 to (2**addrsize)-1 loop
                  if (block_readi(i) = '0') and (block_readf(i) = '
                  0') then
                  -- se comprueba la proteccion contra lectura
                        if iniaddress(i)= avalon_addr then
                              indice :=i;
                              coincidencia := coincidencia or '1';
                        else
                              coincidencia := coincidencia or '0';
                        end if;
                  else
                        coincidencia := coincidencia or '0';
                  end if;
            end loop;
            if  coincidencia = '1' and write = '1' then
                  siguiente <= rutina;
                  pos_intervalo <= indice;
                  pos_en <= '1';
                  block_write <= '1';
                  block_en <= '1';
            else
                  siguiente <= espera;
```

```
		end if;
when rutina =>
		enable <= '1';
		if watchdog < unsigned(timer) then
					-- comprobacion del watchdog
			if avalon_addr = outaddress and write = '1' then
				siguiente <= fin_rutina;
			else
				siguiente <= rutina;
			end if;
		else
					-- error de ejecucion - limite de tiempo
			siguiente <= espera;
			error_spy <= '1';
			clear <= '1';
			checki <= '0';
			block_en <= '1';
		end if;
when fin_rutina =>
		enable <= '1';
		if checki= '0' then
			siguiente <= iteracion_1;
		else
			siguiente <= iteracion_2;
		end if;
when iteracion_1 =>
					-- toma el resultado del espia y lo guarda en el banco
					de registros
		clear <= '1';
		wresult <= '1';
		result_regbank <= result_spy;
		checki <= '1';
		siguiente <= espera;
when iteracion_2 =>
					-- toma el resultado del banco de registros y lo
					compara con el del espia
		checki <= '0';
		block_en <= '1';
		if (result_spy = firstresult) then
					-- correcto, ambas iteraciones coinciden
			error_spy <= '0';
```

```
                else
                                        -- error, las ejecuciones han sido distintas
                                        error_spy <= '1';
                end if;
                        clear <= '1';
                        siguiente <= espera;
        end case;
end process;

process(clk,rst)
        begin
                if rst='0' then
                        actual <= espera;
                        pos <= 0;
                        watchdog <= to_unsigned(0,32);
                for i in 0 to (2**addrsize)-1 loop
                        check(i) <= '0';
                end loop;
                elsif clk'event and clk = '1' then
                        actual <= siguiente;
                        if actual = espera then
                                watchdog <= to_unsigned(0,32);
                        elsif actual = rutina then
                                watchdog <= watchdog + to_unsigned(1,32);
                        end if;
                                if pos_en = '1' then
                                        pos <= pos_intervalo;
                                end if;
                                check(pos)<=checki;
                        end if;
   end process;

  process(clk)
        begin
                assert (error_spy = '0')
                        report "---Error de ejecucion---"
                                severity note;
  end process;

  raddress <= pos_intervalo;
  err_spy        <= error_spy;
end control;
```

B.6 Spy.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity spy is

port (
        enable          : in std_logic;
                                -- habilitacion, en el rango de direcciones de algun
                                bloque que se debe espiar
        clear           : in std_logic;
                                -- borra el resultado, indicado por control
        rst             : in  std_logic;
        clk             : in  std_logic;
        newdata         : in std_logic;
                                -- nuevo dato en el bus
        data            : in std_logic_vector(31 downto 0);
                                -- bus de datos, proveniente de la interfaz
        resultado       : out std_logic_vector(31 downto 0)
                                -- se pasa el resultado al control
        );
end;

architecture spy of spy is

-- registros del modulo
        signal result: std_logic_vector (31 downto 0);
                                -- total acumulado
        constant poli_prim : unsigned (32 downto 0) :=
        "10001100000000000000000000000011";

begin

        process (clk,rst)
        begin
                -- reset
                if rst = '0' then
                        result <= (others => '0');
```

```
                elsif clk'event and clk = '1' then
                        if clear = '1' then
                                                -- inicializacion del resultado, tiene preferencia
                                                clear sobre enable
                                result <= (others => '0');
                        elsif enable = '1' then
                                                -- lectura del dato del bus y actualizacion del
                                                resultado
                                if newdata = '1' then
                                                -- generacion de la firma
                                        result(0) <= data(0) xor (result(31) and
poli_prim(32));
        for i in 1 to 31 loop
        result(i) <= data(i) xor result(i-1) xor (result(31) and poli_prim(32-i));
        end loop;
                                end if;
                        end if;
                end if;
        end process;
        -- salida del resultado
        resultado <= result;
end spy;
```